UNIVERSIDAD NACIONAL DE LA MATANZA

DEPARTAMENTO DE INGENIERÍA E INVESTIGACIONES TECNOLÓGICAS

Proyecto Código C 141

PROGRAMA PROINCE

DATA MINING Y SIMULACIÓN EN EVALUACIONES DE BIODIVERSIDAD

Director: Mg. Santa María, Cristóbal Raúl
Integrantes:
Dr. Soria, Marcelo A.
Ing. López, Luis
Ing. Martínez, Pablo
Sr. Cacho Mendoza, Ariel
Fecha de Inicio: 2013/01/01
Fecha de Finalización: 2014/12/31

Resumen

La investigación realizada se refiere a mediciones de biodiversidad en relevamientos metagenómicos. La secuenciación de ADN permite contar con información procedente de una comunidad microbiológica en cantidad suficiente como para intentar establecer parámetros de riqueza y diversidad de taxones de la población. A partir de una muestra de secuencias de ADN de un gen marcador es posible inferir la cantidad de especies, o en general taxones, presentes en la comunidad y su distribución. Esta inferencia ofrece dificultades que el trabajo ha intentado superar. Es usual que la inferencia estadística de la riqueza biológica por técnicas no paramétricas subestime los valores reales que presenta la población. Se propone entonces un algoritmo de recuento de especies (ARE) que mejora sensiblemente la estimación Ante la imposibilidad fáctica, tecnológica y económica, de contar con secuencias correspondientes al total de una población, el testeo de los resultados obtenidos se realizó construyendo una población simulada basada en una distribución de Fischer de las especies. También se tomaron submuestras de una muestra de gran tamaño cuya riqueza real era conocida y se testeo la eficiencia del algoritmo bajo tal situación. Finalmente el proyecto abordó la programación y paralelización del procedimiento ARE en lenguaje C lo que abrevió los tiempos de ejecución requeridos por los programas prototipo que habían sido realizados con software R. Se considera haber obtenido un procedimiento que constituye una forma sustancialmente mejorada de evaluación de riqueza para las comunidades microbiológicas, a la vez eficiente también en su cálculo computacional y con sensible impacto en el ámbito de los estudios de biodiversidad referida a microorganismos presentes en suelos, atmósfera o en el medio interno humano y animal.

Palabras Clave: metagenómica, riqueza, inferencia, algoritmo, software
Área de conocimiento: Computación
Código de Área de conocimiento: 1802
Disciplina de conocimiento: Métodos Numéricos y Computación
Código Disciplina de conocimiento: 0706
Definición de campo de Aplicación: Biología
Campo de Aplicación: Microbiología
Código Campo de Aplicación: 0214

Data Mining y Simulación en Evaluaciones de Biodiversidad

El trabajo se propuso evaluar el desempeño del Algoritmo de Recuento de Especies (ARE) sobre comunidades microbianas reales para establecer ajustadamente sus propiedades y limitaciones a la vez que desarrollar un software que ejecutara el algoritmo en tiempo óptimo y resolviera las demoras planteadas por la programación piloto realizada en lenguaje estadístico R. En relación con el primer objetivo se construyó una población simulada, distribuida según Fischer, y se tomaron luego muestras de la misma para comparar la evaluación de riqueza aportada por ARE con la riqueza real conocida y con la estimada por procedimientos no paramétricos. Respecto de la programación se desarrolló un código en lenguaje C al que se incorporó también una rutina de generación de números aleatorios adecuada. Se construyó una interface para la carga de los archivos con las muestras y de los distintos parámetros de interés para el procedimiento. Se realizó también una paralelización del algoritmo ARE para que se ejecutaran en simultáneo varias corridas sobre la misma o distintas muestra de secuencias de ADN. Los resultados obtenidos demostraron un desempeño significativamente superior de ARE respecto de las evaluaciones no paramétricas. Todas estas tareas motivaron participaciones en congresos científicos y la presentación de un artículo a la revista Bioinformatics para su evaluación y eventual publicación.

1. INTRODUCCIÓN

Esta investigación es continuidad de los proyectos C96 y C112 referidos a mediciones de biodiversidad en relevamientos metagenómicos. Las nuevas técnicas de secuenciación de ADN permiten contar con información procedente de una comunidad microbiológica en cantidad suficiente como para intentar establecer parámetros de riqueza y diversidad de taxones en la misma. Con los procedimientos de alineado, filtrado y agrupamiento de secuencias en "clusters" denominados Unidades Taxonómicas Operacionales (OTUs) se puede conocer la cantidad de taxones, tales como especies o familias, presentes en una muestra de una comunidad. A partir de esto es posible inferir la cantidad de especies presentes en la población total y su distribución. Sin embargo varios aspectos de esta inferencia ofrecen dificultades aún no suficientemente resueltas. En particular la inferencia estadística de la riqueza biológica por técnicas no paramétricas, medida por ejemplo en cantidad de especies, subestima los valores reales que presenta la población. Respeto a este inconveniente el Mg. Cristóbal Santa María y el Dr. Marcelo Soria desarrollaron en el marco del Proyecto C112 un algoritmo de recuento de especies (ARE) para intentar mejorar la estimación. Así la programación desarrollada en lenguaje R permitió realizar pruebas cuyos resultados arrojaron estimaciones que estaban más cercanas a los valores reales de riqueza sugeridos por criterios biológicos cualitativos. Sin embargo, en esos casos analizados en el proyecto C112, ante el desconocimiento cuantitativo de la riqueza real no era posible evaluar en forma adecuada la mejoría producida. También se hacía evidente la necesidad de ajustar la programación de estos procedimientos para abreviar sustancialmente los tiempos de ejecución de los programas prototipo. A efecto de llevar adelante el trabajo se decidió suponer además que los posibles efectos sobre las estimaciones de riqueza de las técnicas de agrupamientos de secuencias en OTUs habían sido adecuadamente tratados en forma previa. La investigación se focalizó en los siguientes objetivos: i) Construir nuevos desarrollos de simulación de poblaciones y testear el algoritmo ARE afinando su desempeño ii) Reprogramar este algoritmo dotándolo de mayor eficiencia computacional. Tales objetivos de trabajo se realizaron bajo la hipótesis principal que la estimación de la riqueza poblacional en comunidades microbiológicas resulta sustancialmente mejorada por el procedimiento ARE con eficiente cálculo computacional. Comprobar tal hipótesis tendría impacto en el ámbito de los estudios de biodiversidad referida a microorganismos presentes en medios tales como suelos, atmósfera, interno humano y animal, etc. El presente informe final dará cuenta de las tareas realizadas y los resultados alcanzados al trabajar en cada uno de los objetivos.

2. CONTEXTO

Cuantificar la biodiversidad en comunidades microbianas resulta una compleja tarea pues, además de presentarse problemas estadísticos parecidos a los que se hallan cuando hay que inferir riqueza o diversidad en otro tipo de poblaciones (Magurran, 2011), se agregan al caso sesgos debidos al proceso del ADN metagenómico a partir del cual se identifican los distintos individuos (Schloss 2010), (Youssef, N and Elshahed, M. 2008), (Huse et al. 2010). Este trabajo se ciñe a explorar una metodología experimental para tratar la inferencia estadística de la riqueza poblacional y supone resueltos, o al menos atenuados por apropiados recaudos, los efectos producidos por la secuenciación, el alineamiento, el filtrado y cualquier otro proceso del ADN obtenido a partir de una muestra de la comunidad.

La técnica utilizada para las mediciones de riqueza microbiana requiere un gen marcador, de alta conservación a través de la evolución (Schloss-Handelsman 2006). Así, cuando las secuencias pertenecientes a individuos de una misma muestra revelan un cierto porcentaje de variaciones entre ellas, tales variaciones pueden

atribuirse a la diferencia de especies, y en general de taxones, y no a simples ocurrencias aleatorias.

Transformada la muestra en un conjunto de secuencias del gen marcador cabe inferir a través de ella la riqueza de la comunidad y quizás también su distribución. Es aquí donde aparece un problema estadístico importante pues lo usual es que el tamaño muestral no sea suficiente para la tarea y en los hechos la riqueza sea subestimada (Hughes et al., 2001) Esto se debe a la presencia de una proporción importante de especies o taxones raros, en términos estadísticos, lo que hace muy poco probable que se encuentren en la muestra individuos que pertenezcan a todas o casi todas ellas. Los individuos de especies raras son muy pocos en relación con los individuos de especies abundantes en la comunidad y además las especies raras son mucho más que las abundantes por lo cual el tamaño de la muestra debería ser muy grande para inferir un valor razonablemente aproximado de la riqueza. Es decir; la rareza y la distribución de especies dificultan seriamente la estimación de la riqueza poblacional (Roesch, L. et al. 2007). Este problema se ha intentado resolver a través de la construcción de estimadores no parámetricos como CHAO (Chao 1984) y ACE (Chao-Lee 1992) que si bien han mejorado las estimaciones no han dado por resuelta la inferencia, al menos cuando se trata de poblaciones microbianas.

Una línea de trabajo actual (Haegeman et al., 2013) considera que la evaluación apropiada de la biodiversidad requiere el análisis de un conjunto de índices de Hill que representan riqueza, entropía etc. (Hill 1973) O´Hara, R. 2005) Esta perspectiva se amplía con el desarrollo de la extrapolación de las curvas de rarefacción para evaluar la riqueza (Chao et al. 2014). El presente trabajo propone un procedimiento alternativo a los mencionados al construir un proceso aleatorio que va incrementando en forma simulada el tamaño muestral utilizando una estimación de la probabilidad de que, dada una muestra de tamaño *n*, un próximo individuo que se agregue a ella corresponda a una especie nueva. Esta estimación fue comunicada por Alan Turing en 1941 (Good 1953), (Nadas 1985).

Así el Algoritmo de Recuento de Especies (ARE) diseñado actualiza la cantidad de individuos y especies en cada iteración y va haciendo crecer el número de especies en simultáneo con la disminución hacia cero de la probabilidad de hallar una nueva especie. Si bien la distribución resultante puede no corresponderse estadísticamente con la real de la población, las pruebas realizadas a partir de muestras de poblaciones simuladas y reales permiten ver que la riqueza, atenuado todo sesgo por procesamiento previo de las secuencias, resulta mejor apreciada. Este procedimiento de modelado experimental de la comunidad se hace computacionalmente posible y razonable al optimizar los tiempos de ejecución y contar con capacidades de procesamiento en paralelo.

3. MATERIALES Y MÉTODOS

3.1- En metagenómica se analizan las propiedades de una comunidad a partir de los genomas de los individuos que la integran. En particular, dada una comunidad microbiana es posible tomar una muestra y procesar el ADN presente para obtener secuencias del gen marcador elegido que, en el presente trabajo, es el 16S rRNA (Armougom and Raoult 2009). Cada secuencia de este gen corresponde entonces a un individuo distinto que integra la muestra y resulta posible agrupar las secuencias de acuerdo a criterios de proximidad evaluados por el modelo de distancia genética de Jukes-Cantor u otro similar (Hillis et al. 1996). Puede tomarse entonces un umbral de disimilaridad entre secuencias a partir del cual se las considerará especies (o más generalmente taxones) diferentes. Así se forman distintos clusters que representan cada uno a una especie (o taxón) integrado por individuos que forman la muestra y son similares de acuerdo al umbral elegido (Schloss-Handelsman 2005) Las *n* secuencias originales de la muestra quedan distribuidas en los clusters denominados Unidades Taxonómicas Operacionales y puede construirse entonces la distribución de

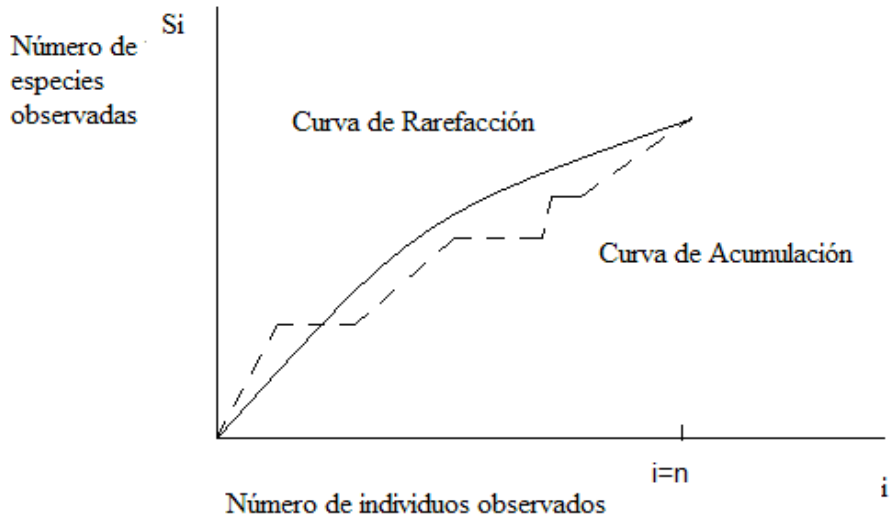
abundancia muestral que indica cuantos clusters contienen *r* individuos, siendo $r = 1,2,3,..n$. (Hill et al. 2003)

Al seleccionar aleatoriamente *n* individuos de una comunidad que contiene un número finito y desconocido de especies *S*, las cantidades $n_1, n_2, ..., n_r, ..., n_k$ son, en cada caso, el número de especies que contabilizan *r* individuos entre los *n* seleccionados. De tal forma $\sum_{r=1}^{k} r n_r = n$ Es claro que n_0 expresa la cantidad de especies que no aparece representada por ninguno de los *n* individuos elegidos y que la cantidad de especies en la comunidad es $S = \sum_{r=0}^{k} n_r$ Si se denomina S_n a la cantidad de especies contabilizadas al tomar *n* individuos resulta $S_n = S - n_0$ (Chao and Shen. 2003) Hay que observar que al agregar un nuevo individuo a una muestra, éste puede resultar perteneciente a una especie ya presente o a una nueva aún no contabilizada.

3.2- El método de las curvas de rarefacción permite estimar la riqueza de un medio aunque generalmente es aplicado para comparar riqueza entre dos o más comunidades, pues alivia los problemas derivados del tamaño insuficiente y habitualmente desigual de las muestras (Hughes-Hellmann 2005) (Gotelli and Colwell. 2001) La idea básica de la rarefacción por individuos al tratar de estimar la riqueza es que a partir de la toma de muestras de mayor tamaño será posible capturar un número creciente de especies distintas. Dada una comunidad que tenga una cantidad desconocida de individuos y un número S de especies distintas también desconocido, se pueden tomar muestras de tamaño n y determinar S_n que es el número de especies distintas halladas en la muestra. Ante cada reordenamiento que se efectúe en la muestra, al ir examinando cada individuo, el número de especies detectadas irá creciendo acumulativamente según la curva punteada de la Figura 1. Contabilizados *i* de los *n* individuos en cada reordenamiento, el valor esperado teórico de $E(S_i)$ se aproxima por el promedio de los S_i obtenidos para cada uno de ellos. La curva formada por los puntos $(i, E(S_i))$, que está dibujada en trazo continuo en la Figura 1, se denomina de rarefacción y puede utilizarse para estimar la riqueza S del medio.

Figura 1

Acumulación y Rarefacción



La curva de rarefacción representa el promedio de todas las curvas de acumulación construidas para los distintos remuestreos. Cuando se alcanza la cantidad *i=n* de individuos examinados, cualquier curva de acumulación habrá contabilizado la cantidad S_n de especies presentes en la muestra de ese tamaño y por lo tanto $E(S_n) = S_n$. Si además *n* resultara suficientemente grande, la curva de rarefacción tendería en forma asintótica al valor de la riqueza poblacional S. Es decir; para determinar la asíntota horizontal de una curva de rarefacción debe aumentarse el tamaño muestral hasta que $E(S_n)$ observe un comportamiento constante al crecer *n* y en tal circunstancia la cantidad $E(S_n) = S_n$ aproxima a la riqueza poblacional S.

3.3- Se construye un modelo experimental a través de un proceso aleatorio definido de la siguiente manera.

Definición 1: Dada una muestra de tamaño *n* sea, para cada i, con $i = 1,2,3,...$, la variable aleatoria S_i que toma los valores $S_i = S_{i-1}$ y $S_i = S_{i-1} + 1$ con probabilidades respectivas $1 - p_i$ y p_i siendo además $S_0 = S_n$. La sucesión de variables aleatorias $S_1, S_2, S_3, ...$ se denomina en adelante Proceso Aleatorio de Cantidad de Especies (Figura 2).

Figura 2

Proceso Aleatorio de Cantidad de Especies

…	S_{i-2}	S_{i-1}	S_i	S_{i+1}	S_{i+2}	…
…	i-2	i-1	i	i+1	i+2	…

La interpretación del modelo experimental identifica a S_i como la cantidad de especies distintas presentes en una muestra de tamaño *n+i*. Además p_i se interpreta como la probabilidad de que al incorporar un nuevo individuo a una muestra de tamaño n+ $i-1$, éste corresponda a una especie nueva no presente hasta ahora en la muestra. Se prueba (Good 1953) que la probabilidad de que, elegidos *n* individuos, al seleccionar uno nuevo éste resulte de una especie hasta ahora no contabilizada,

puede aproximarse por el cociente $T = \frac{n_1}{n}$ dónde n_1 es el número de especies que aparece una vez en la muestra elegida y que debe suponerse mayor estricto que 0. Esta idea, aportada por Turing, es la que aquí se utiliza para estimar p_i. Se propone entonces $T_i = \frac{n°sgletons}{n+i-1}$ como fórmula de cálculo de la probabilidad de especie nueva asociada al proceso aleatorio de cantidad de especies (Tabla 1).El número de singletons en cada muestra de tamaño $n+i-1$ refiere a los clusters formados por un solo individuo, cuando se realiza el procedimiento de agrupamiento de secuencias y la consiguiente identificación de cada cluster con una especie distinta.

Tabla 1

Probabilidad de Estado

Estado	Probabilidad de estado
$S_i=S_{i-1}$	$p_i = 1 - T_i$
$S_i=S_{i-1}+1$	$p_i = T_i$

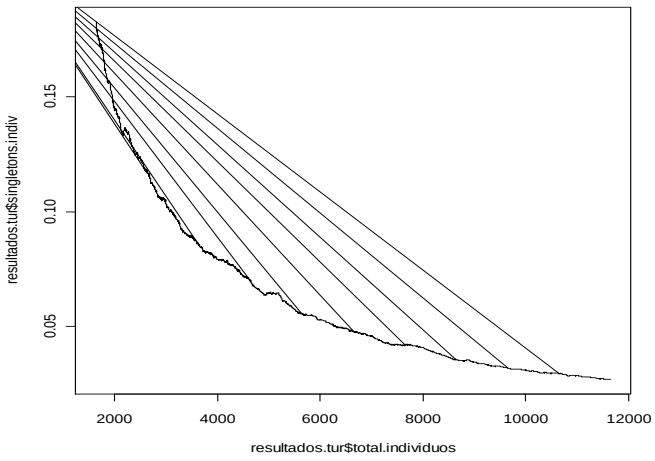
Así el valor esperado de S_i, que correspondería a la curva de rarefacción cuando se han agregado i individuos simulados a la muestra es:

$E(S_i) = S_{i-1}(1-T_i) + (S_{i.-1}+1)T_i$ y operando se obtiene $E(S_i) = S_{i-1} + T_i$

La Figura 3 muestra como la probabilidad de especie nueva tiende a cero, $T_i \to 0$, conforme va creciendo la cantidad de individuos incorporados a la muestra mediante la simulación de acuerdo al modelo experimental. Al ser finita la cantidad buscada de especies S (Good 1953), la curva de rarefacción $E(S_i)$ debe ir alcanzando una asíntota que la estime.

Figura 3

Disminución de la Probabilidad de Especie Nueva



Hay que considerar que el valor de *T* en cada iteración es en realidad una estimación de la probabilidad de especie nueva y no exactamente esa probabilidad. Por esta razón el valor esperado $E(S_i)$ resultará solo una estimación aproximada de S. De tal forma el modelo construido resultará adecuado si la experiencia computacional muestra un buen desempeño en la evaluación de S. En la práctica, con

la idea de reducir la varianza, a partir de la misma muestra el cálculo podrá realizarse varias veces y promediar las cantidades estimadas.

3.4- El Algoritmo de Recuento de Especies (ARE) se basa en el Proceso Aleatorio de Cantidad de Especies antes definido. Realiza la simulación por la técnica de Monte Carlo. Dado el tamaño n de la muestra original se determina el valor del estimador de la probabilidad de especie nueva. Ese valor permite constituir los intervalos $[0,T_n]$ y $(T_n,1]$ de modo que al elegir un número aleatorio r tal que $0 \le r \le 1$, si cae dentro del primer intervalo el nuevo individuo simulado corresponda a una especie nueva y si cae dentro del segundo intervalo sea un ejemplar de una especie conocida. Si ocurre lo primero, la cantidad de especies en el medio se incrementa en 1 y si no, se utilizan las proporciones existentes de cada especie para asignar por medio de un nuevo número aleatorio la especie ya conocida, a la cual pertenece el nuevo individuo. Así se van agregando individuos hasta que la cuenta de las especies nuevas alcance un valor estable o hasta que se cumpla con algún otro criterio de corte de la simulación. Los pasos del procedimiento se sintetizan en forma secuenciada a continuación.

i- Dada la muestra elegida, de tamaño n, y su agrupamiento en OTUs, se determina el valor inicial del estimador de Turing $T_{i+1} = \frac{f_1}{i}$ siendo $i = n$ y f_1 el número actual de singletons.

ii- Se elige un número aleatorio r, tal que $0 \le r \le 1$ y se pregunta si está en el intervalo $[0,T_{i+1}]$. Si es así, se realiza $S_{i+1} = S_i + 1$ y se va al paso iv. Si ocurre lo contrario se realiza $S_{i+1} = S_i$ y se va al paso iii

iii- Se utiliza la distribución de abundancia de la muestra para calcular la proporción de individuos que están en OTUs de $1,2,..n$ individuos y con estas proporciones se determina, por un sorteo de acuerdo a ellas, a que grupo de OTUs ya conocidas pertenece el nuevo individuo. Para establecer a que OTU específica, de entre las de este grupo, corresponde el nuevo individuo se realiza un nuevo sorteo con probabilidad uniforme para cada OTU del grupo.

iv- Sea el nuevo individuo de una nueva especie o no, la muestra tiene ahora un elemento más. Se pregunta entonces si el procedimiento debe cortarse porque se cumple el criterio elegido para ello en cuyo caso la simulación ha finalizado. Si el criterio de corte no se cumple, se asigna entonces $i \leftarrow i+1$, se calcula la nueva distribución de abundancia y la nueva estimación de Turing según $T_{i+1} = \frac{f_1}{i}$ y se repite desde el paso ii.

El programa de computadora correspondiente fue desarrollado en lenguaje R y se presenta como anexo.

3.5- No es posible contar con secuencias del gen 16S rRNA correspondientes a todos los individuos de una comunidad microbiana real pues, además de limitaciones tecnológicas o económicas, el ensayo resultaría destructivo de la propia comunidad. Por esta razón, para realizar una prueba del procedimiento ARE sobre una población completa, cabe simularla o trabajar con una muestra para la cual la curva de rarefacción haya alcanzado la asíntota que estima la cantidad de especies. En ambos casos se puede dar entonces por conocida la real cantidad de especies y compararla con la estimación que surge de aplicar el procedimiento ARE. Esto también permite comparar el desempeño de ARE con otras formas de estimación.

Con tal idea se construyó una comunidad simulada utilizando la serie log $\alpha x, \frac{\alpha x^2}{2}, \frac{\alpha x^3}{3}, \ldots, \frac{\alpha x^m}{m}$ que modela la cantidad esperada de especies que están representadas por 1, 2,…, *m* individuos en la población. Si *N* es el tamaño de la población y *S* la cantidad de especies en ella contenida, se cumplen las relaciones $\frac{S}{N} = [(1-x)/x][-\ln(1-x)]$ y $\alpha = \frac{N(1-x)}{x}$ (Fischer et al. 1943). De acuerdo a esto pueden calcularse valores de *S* y *N* a partir de cantidades *α* y *x* de significación ecológica. Se utilizaron entonces los valores *α= 5000* y x=0.995 (Magurran 2004) para obtener una comunidad integrada por 898341 individuos distribuidos entre 26332 especies.

Por otro lado, para trabajar también sobre datos reales, se consideró la muestra de agua de mar profundo FS396,archea (Haegeman et al. 2013) y (Huber et al. 2007) integrada por 16316 secuencias de 16S rRNA y cuya curva de rarefacción alcanza un comportamiento asintótico con ese tamaño. La cantidad de especies presentes en esta muestra, que a efecto de la experiencia se consideró como una comunidad completa, es 346.

En ambos casos la cantidad observada de especies se comparó con las estimaciones obtenidas por los estadísticos no paramétricos CHAO y ACE y a su vez éstos con las estimaciones realizadas por ARE. Para la muestra obtenida de la población simulada se estudió el desarrollo de la estimación conforme aumentara la cantidad de individuos simulados y disminuyera a su vez el valor del estimador de Turing. Para la muestra FS396,archea se tomaron 5 submuestras de 6 tamaños diferentes y con cada una de ellas se realizaron 10 corridas del algoritmo a efecto de estimar el valor esperado $E(S_i)$ que a su vez estima S. A la comparación de desempeño se le agregó la evaluación de la extrapolación realizada por medio de

$$\hat{S}_{n+m} \approx S_n + \hat{f}_0 \left[1 - \left(1 - \frac{f_1}{n\hat{f}_0 + f_1} \right)^m \right] \approx S_n + \hat{f}_0 \left[1 - \exp\left(\frac{-mf_1}{n\hat{f}_0 + f_1} \right) \right]$$ (Chao et al. 2014)

Aquí S_n es la riqueza de la muestra de tamaño *n* inicial que se calcula a partir de la misma, f_1 es la cantidad de singletons en la muestra de tamaño *n* y $\hat{f}_0$ es una estimación de la cantidad de especies no observadas en la muestra de tamaño *n*. Por valor $\hat{f}_0$ se tomó la estimación ACE que en realidad está pensada como estimador del total de especies en la comunidad y no como estimador de las especies faltantes en la muestra. De tal forma esta última cantidad puede incluso resultar sobrevaluada. *m* es la cantidad de individuos que se agregan idealmente para extrapolar. En el caso en que *m* se hiciera muy grande ($m \to \infty$) resultaría $\hat{S}_{n+m} \approx S_n + \hat{f}_0$ con lo que tal valor podría tomarse como una cota superior de la extrapolación que en principio también se considerará en el análisis. Por último se realizó una evaluación de los respectivos coeficientes de variación.

3.6- Durante la ejecución del proyecto C112 se realizó la programación del algoritmo ARE en lenguaje R a efecto de realizar las pruebas. La lentitud de ejecución del programa crecía con el tamaño de las muestras. Si bien estaba operativo, requería para su uso en experimentos reales mayor velocidad de ejecución y más opciones de procesamiento. Esto motivó la idea de desarrollar un software que tuviera todas las funcionalidades necesarias a la vez que acortara los tiempos de proceso.

La muestra con las secuencias de ADN correspondientes al gen marcador agrupadas en Unidades Taxonómicas Operacionales está contenida en una planilla de cálculo donde cada columna representa una OTU distinta y cada fila corresponde a un

nivel de disimilaridad diferente. Así cada celda de la planilla contiene las secuencias que corresponden a una OTU establecida según un nivel de disimilaridad, en particular identificadas con un código predeterminado. Tal planilla es la salida usual del programa de procesamiento de secuencias MOTHUR, de caracter libre, que realiza previamente la lectura, el alineado, el filtrado y el clustering de las secuencias genéticas obtenidas desde repositorios internacionales o a partir de una secuenciación propia. Esta planilla se identifica con la extension final .list y resulta el insumo básico con el que debe alimentarse el programa que ejecuta el algoritmo ARE. La Figura 4 muestra como ejemplo un porción del archivo S86.phy.cgi.fn.list obtenido por secuenciación y procesamiento con MOTHUR a partir de muestras del suelo de La Sal del Rey, región lacustre hipersalina de baja profundidad, ubicada en el Estado de Texas, EEUU, cuyas coordenadas son (26° 31’ 55’’ N, 98° 03’ 50’’ O). (Hollister et al. 2010)

Figura 4

unique	7293	S86-3191,S86	S86-2	S86-3	S86-2001,S86	S86-270,S86-	S86-6	S86-7	S86-8	S86-713,S86-	S86-10	S86-11	S86-12
0.00	6617	S86-3191,S86	S86-2	S86-3	S86-2001,S86	S86-270,S86-	S86-6	S86-7	S86-8	S86-713,S86-	S86-10	S86-11	S86-12
0.01	5360	S86-3191,S86	S86-2	S86-3	S86-2001,S86	S86-270,S86-	S86-6	S86-7	S86-8	S86-713,S86-	S86-10	S86-11	S86-12
0.02	4688	S86-3191,S86	S86-2	S86-3	S86-2001,S86	S86-270,S86-	S86-6	S86-7	S86-8	S86-713,S86-	S86-10	S86-11	S86-12
0.03	4245	S86-3191,S86	S86-2	S86-3	S86-2001,S86	S86-270,S86-	S86-6	S86-7	S86-8	S86-713,S86-	S86-10	S86-11	S86-12
0.04	3893	S86-3191,S86	S86-2	S86-3	S86-2001,S86	S86-270,S86-	S86-6	S86-7	S86-8	S86-713,S86-	S86-10	S86-11	S86-12
0.05	3575	S86-3191,S86	S86-2	S86-3	S86-2001,S86	S86-270,S86-	S86-6	S86-7	S86-8	S86-713,S86-	S86-10	S86-11	S86-12

La primera columna corresponde a los distintos umbrales de disimilaridad y cada una de las restantes representa un cluster (OUT) diferente . cada celda contiene entonces los códigos de los individuos contenidos en ese cluster.

De acuerdo a lo expresado la primer tarea del programa debe ser la lectura de uno o más de estos archivos.

A continuación esta información debe usarse para correr el algoritmo ARE seleccionando el 'Umbral de Disimilitud' que se aceptará para que las secuencias integren las distintas OTUs y la forma de finalizar la ejecución según el valor que alcance T, el estimador de Turing de la probabilidad de nueva especie, o bien una vez que se haya simulado la incorporación de una cantidad fija de individuos. También se debe seleccionar la cantidad de veces que se realizará la simulación utilizando la misma muestra y finalmente hay que seleccionar los 'Hilos de Ejecución' en paralelo que permitan trabajar en simultáneo con distintas muestras.

Para cada simulación el programa deberá ejecutar la parte propia de ARE. Ejecutadas las simulaciones requeridas el software habrá de hacer visibles los resultados obtenidos para su evaluación. En particular deberá mostrar, para cada corrida, la cantidad de especies o en general taxones estimados, la cantidad de simulaciones realizadas y el valor alcanzado por T. A su vez deberá resumir esta información en forma promediada para cada muestra procesada. La descripción técnica se realiza a continuación.

4. DESCRIPCIÓN DEL SOFTWARE

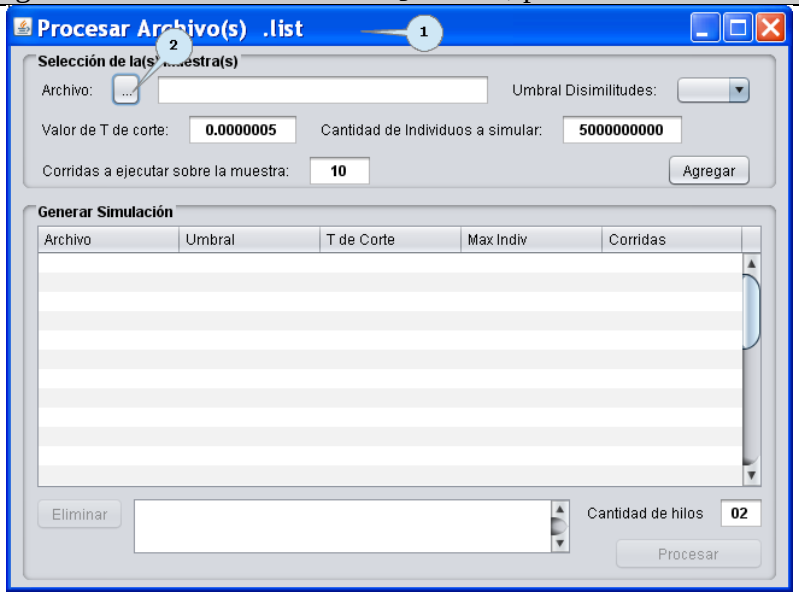
Nota: los números entre paréntesis refieren a la pantalla del programa que se muestra más abajo.

Se desarrolló el programa interactivo 'LeerList.jar' (1) que permite seleccionar (2) uno o múltiples archivos generados por el secuenciador (3) y para cada uno de esos archivos (.list) (4) , se puede seleccionar el 'Umbral de Disimilitud' (5) a procesar, el 'T

de corte' (6) y/o la 'Cantidad de Individuos a Generar' (7) (lo que suceda primero en la simulación dará fin a la misma), además de la cantidad de simulaciones o 'Corridas a Ejecutar Sobre Cada Muestra' (8).
Se podrá finalmente indicar la cantidad de 'Hilos de Ejecución' (9), que ejecutará en procesos separados la simulación aprovechando el procesamiento con más de un procesador o núcleos del mismo.
En todo momento, a medida que se seleccionan y se procede a 'Agregar' (10) simulaciones, se las podrá 'Eliminar' (11) de la lista de simulación y agregar más.
Una vez terminada la selección de los archivos se envía a 'Procesar' (12) con lo que el programa comienza por eliminar los archivos temporarios generados por corridas anteriores y a eliminar y volver a generar el directorio temporario ('tmp') en que genera estas salidas temporarias. La Figura 5 muestra la pantalla interactiva en los distintos pasos.

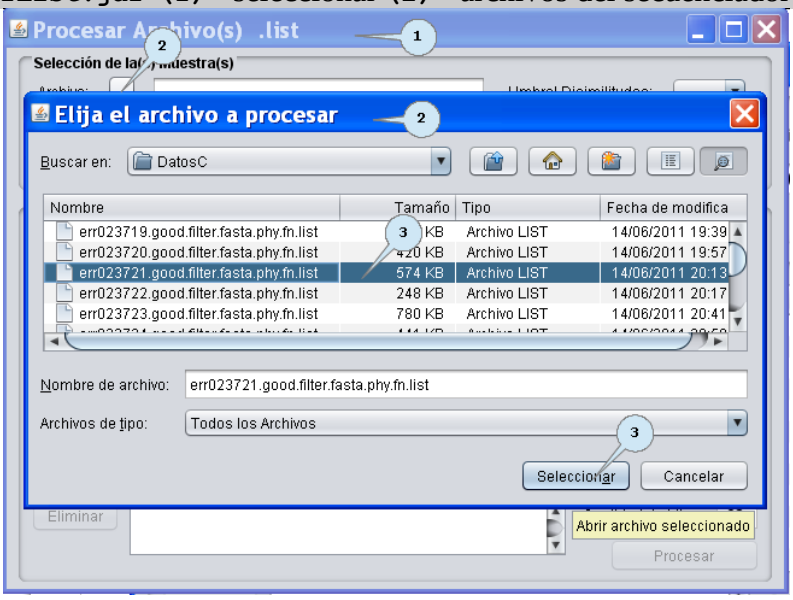
Figura 5

Programa interactivo '**LeerList.jar**' **(1)**, permita seleccionar **(2)**



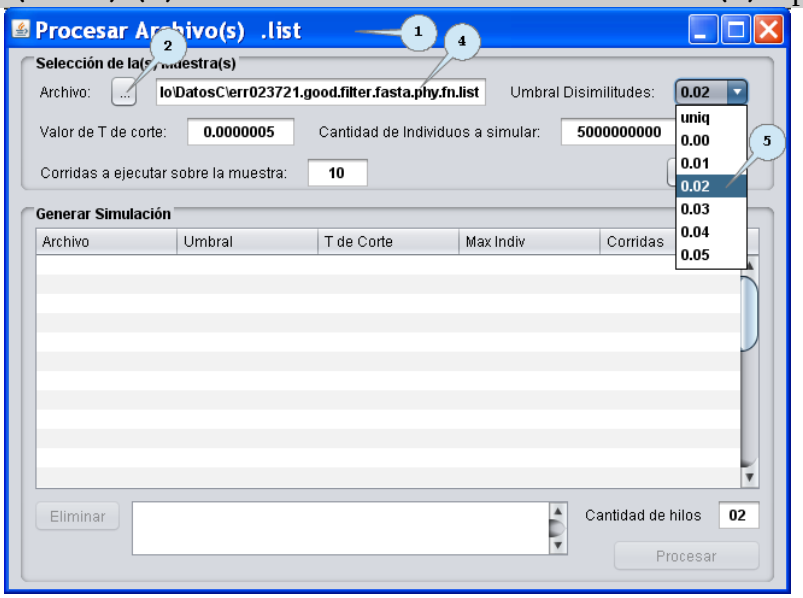
Volver

'**`LeerList.jar`** **`(1)`** - seleccionar **`(2)`** - archivos del secuenciador **`(3)`**



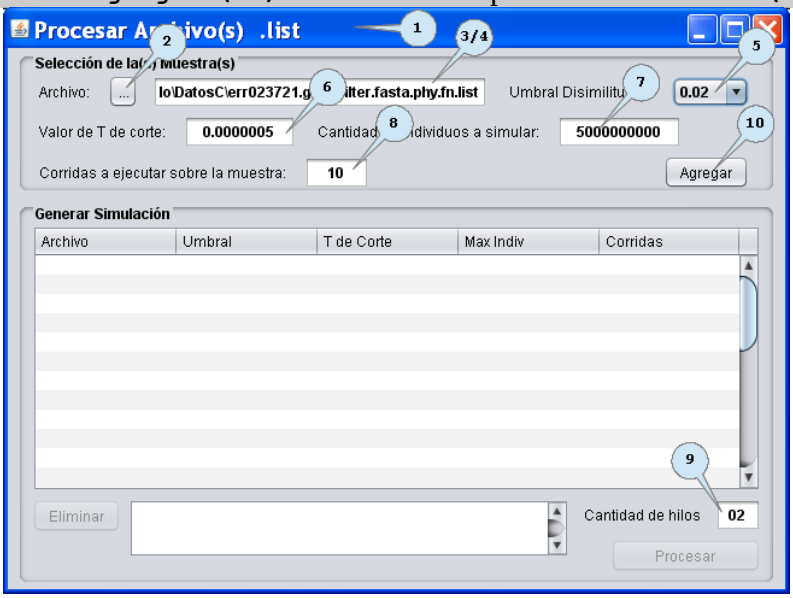
Volver

'**`LeerList.jar`** **`(1)`** - seleccionar **`(2)`** - archivos del secuenciador **`(3)`**
archivos **`(.list)`** **`(4)`** - seleccionar '**`Umbral de Disimilitud`**' **`(5)`** a procesar



Volver

'**LeerList.jar**' **(1)** - seleccionar **(2)** - archivos del secuenciador **(3)**
archivos **(.list)** **(4)** - seleccionar '**Umbral de Disimilitud**' **(5)** a procesar
seleccionar '**Umbral de Disimilitud**' **(5)** - el '**T de corte**' **(6)**
'**Cantidad de Individuos a Generar**' **(7)**
'**Corridas a Ejecutar Sobre Cada Muestra**' **(8).**
indicar cantidad de '**Hilos de Ejecución**' **(9)**
proceder '**Agregar**' **(10)** simulaciones - pudiendo '**Eliminar**' **(11)**

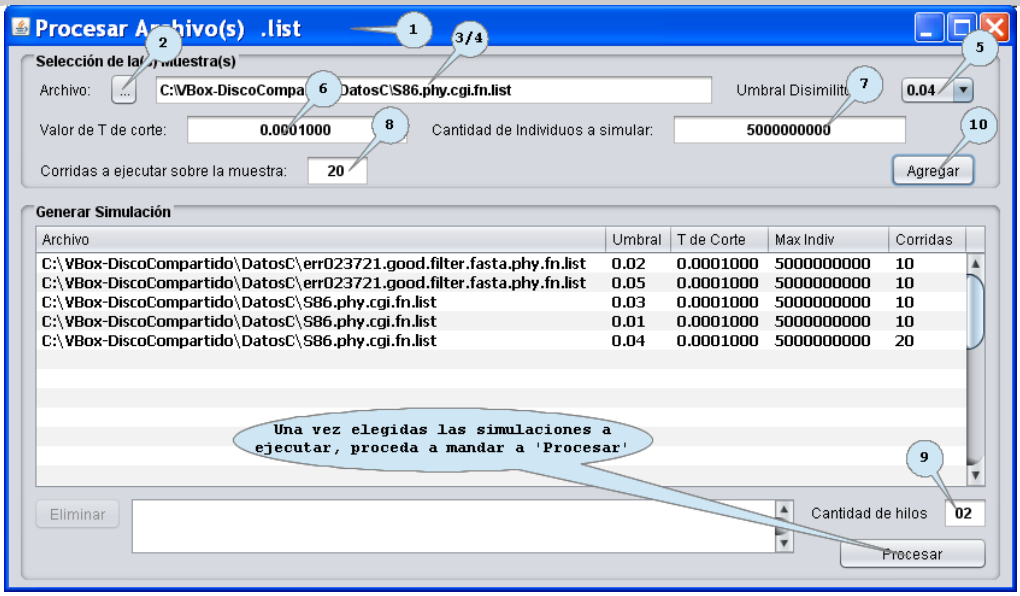


Volver

'**LeerList.jar**' **(1)** - seleccionar **(2)** - archivos del secuenciador **(3)**
archivos **(.list)** **(4)** - seleccionar '**Umbral de Disimilitud**' **(5)** a procesar
seleccionar '**Umbral de Disimilitud**' **(5)** - el '**T de corte**' **(6)**
'**Cantidad de Individuos a Generar**' **(7)**
'**Corridas a Ejecutar Sobre Cada Muestra**' **(8).**
indicar cantidad de '**Hilos de Ejecución**' **(9)**
proceder '**Agregar**' **(10)** simulaciones - podiendo '**Eliminar**' **(11)**
Se pueden redimensionar la ventana y las columnas

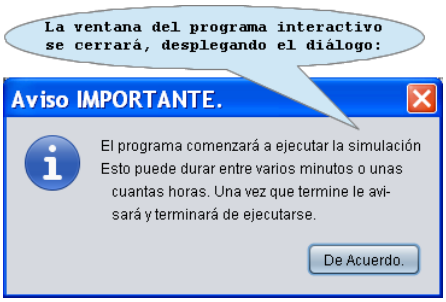
Volver

'**LeerList.jar**' **(1)** - seleccionar **(2)** - archivos del secuenciador **(3)**
archivos **(.list) (4)** - seleccionar '**Umbral de Disimilitud**' **(5)** a procesar
seleccionar '**Umbral de Disimilitud**' **(5)** - el '**T de corte**' **(6)**
'**Cantidad de Individuos a Generar**' **(7)**
'**Corridas a Ejecutar Sobre Cada Muestra**' **(8).**
indicar cantidad de '**Hilos de Ejecución**' **(9)**
proceder '**Agregar**' **(10)** simulaciones - podiendo '**Eliminar**' **(11)**
Se pueden redimensionar la ventana y las columnas
Para finalmente '**Procesar**' lo seleccionado

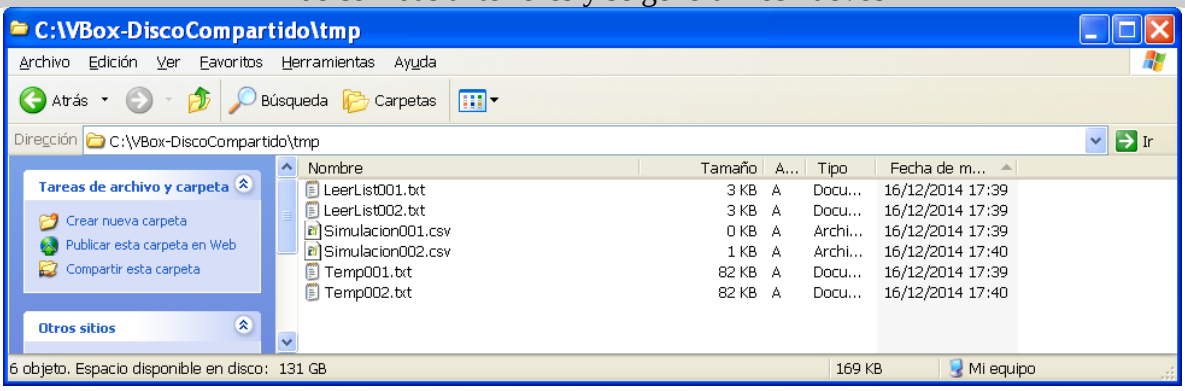


Volver

La ventana interactiva desaparecerá, mostrando un cuadro de diálogo:



En el directorio temporario (tmp), se eliminaron archivos temporarios de corridas anteriores y se generan los nuevos



La ventana interactiva reaparecerá con un nuevo diálogo:

Dando por concluída su tarea

El programa genera tantos archivos 'LeerListddd.txt' como hilos de ejecución se hayan seleccionado, ejecutando el programa 'leerList.exe' , tantas veces como hilos de ejecución, a fin de que haga las simulaciones sobre los archivos indicados en 'leerListddd.txt'

Observación: en adelante ddd representarán los dígitos 001, 002, etc correspondientes al número de hilo de ejecución generado. La Figura 6 muestra el archivos 'LeerListddd.txt' correspondientes a dos hilos de ejecución.

Figura 6

LeerList001.txt - Volver

```
0.02    0.0001000       5000000000      C:\VBox-
DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
0.02    0.0001000       5000000000      C:\VBox-
DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
0.02    0.0001000       5000000000      C:\VBox-
DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
0.02    0.0001000       5000000000      C:\VBox-
DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
0.02    0.0001000       5000000000      C:\VBox-
DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
0.05    0.0001000       5000000000      C:\VBox-
DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
0.05    0.0001000       5000000000      C:\VBox-
DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
0.05    0.0001000       5000000000      C:\VBox-
DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
0.05    0.0001000       5000000000      C:\VBox-
DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
0.05    0.0001000       5000000000      C:\VBox-
DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
0.03    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.03    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.03    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.03    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.03    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.01    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.01    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.01    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.01    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.01    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.04    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.04    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.04    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.04    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.04    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.04    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.04    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.04    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.04    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.04    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
```

Volver

LeerList002.txt - Volver

```
0.02    0.0001000       5000000000      C:\VBox-
DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
0.02    0.0001000       5000000000      C:\VBox-
DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
0.02    0.0001000       5000000000      C:\VBox-
DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
0.02    0.0001000       5000000000      C:\VBox-
DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
0.02    0.0001000       5000000000      C:\VBox-
DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
0.05    0.0001000       5000000000      C:\VBox-
DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
0.05    0.0001000       5000000000      C:\VBox-
DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
0.05    0.0001000       5000000000      C:\VBox-
DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
0.05    0.0001000       5000000000      C:\VBox-
DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
0.05    0.0001000       5000000000      C:\VBox-
DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
0.03    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.03    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.03    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.03    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.03    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.01    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.01    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.01    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.01    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.01    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.04    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.04    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.04    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.04    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.04    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.04    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.04    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.04    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.04    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
0.04    0.0001000       5000000000      C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
```

Volver

Las salidas parciales de cada simulación estarán en los archivos temporarios 'Simulacionddd.csv'. La Figura 7 muestra uno de estos archivos.

Figura 7

.\tmp\Simulacion001.csv

```
       Cant.Clus.:    Singletones:   Tot. Indiv.    T.      T Transc.
Archivo:        C:\VBox-DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
Umb Disimilit.: 0.02
T. de Corte:    0.0001000
Max Individuos: 5000000000
       3177    1845    7637    0.241587       0
       7893    2227    74225   0.030003       0
       10436   2275    227490  0.010000       0
       15453   2189    2188167 0.001000       4
       20954   2355    23432837        0.000100       40
       20975   2359    23590002        0.000100       40
Archivo:        C:\VBox-DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
Umb Disimilit.: 0.02
T. de Corte:    0.0001000
Max Individuos: 5000000000
       3177    1845    7637    0.241587       0
       8057    2238    74606   0.029998       0
       10425   2227    222738  0.009998       0
       15416   2272    2271025 0.001000       4
       20525   2248    22370164        0.000100       38
       20537   2247    22476233        0.000100       38
Archivo:        C:\VBox-DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
Umb Disimilit.: 0.02
T. de Corte:    0.0001000
Max Individuos: 5000000000
       3177    1845    7637    0.241587       0
       7509    2091    69700   0.030000       0
       9760    2057    205691  0.010000       0
       14294   2011    2010682 0.001000       3
       18951   1990    19807279        0.000100       34
       18952   1985    19857657        0.000100       34
Archivo:        C:\VBox-DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
Umb Disimilit.: 0.02
T. de Corte:    0.0001000
Max Individuos: 5000000000
       3177    1845    7637    0.241587       0
       8232    2313    77127   0.029989       0
       10720   2302    230203  0.010000       0
       16192   2375    2374378 0.001000       4
       21289   2209    21980641        0.000100       39
       21302   2212    22128809        0.000100       39
Archivo:        C:\VBox-DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
Umb Disimilit.: 0.02
T. de Corte:    0.0001000
Max Individuos: 5000000000
       3177    1845    7637    0.241587       0
       8658    2482    82733   0.030000       0
       11421   2515    251526  0.009999       1
       17071   2513    2512027 0.001000       4
       22990   2537    25244739        0.000100       45
       22992   2531    25310002        0.000100       45
Archivo:        C:\VBox-DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
Umb Disimilit.: 0.05
T. de Corte:    0.0001000
Max Individuos: 5000000000
       1930    984     7637    0.128846       0
       3323    992     33124   0.029948       0
       4448    1013    101296  0.010000       1
       7116    1113    1113417 0.001000       3
       10115   1256    12497514        0.000100       30
       10116   1254    12540002        0.000100       31
Archivo:        C:\VBox-DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
Umb Disimilit.: 0.05
T. de Corte:    0.0001000
Max Individuos: 5000000000
       1930    984     7637    0.128846       0
       3665    1167    38904   0.029997       0
       4884    1142    114236  0.009997       0
       7642    1173    1172415 0.001000       2
       10334   1160    11550830        0.000100       26
       10342   1160    11605162        0.000100       26
Archivo:        C:\VBox-DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
```

.\tmp\Simulacion001.csv

```
Umb Disimilit.: 0.05
T. de Corte:    0.0001000
Max Individuos: 5000000000
        1930    984     7637    0.128846        0
        3897    1232    41090   0.029983        0
        5370    1267    126757  0.009996        0
        8246    1258    1257373 0.001000        3
        10797   1154    11486897        0.000100        25
        10798   1151    11510002        0.000100        25
Archivo:        C:\VBox-DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
Umb Disimilit.: 0.05
T. de Corte:    0.0001000
Max Individuos: 5000000000
        1930    984     7637    0.128846        0
        3390    1011    33719   0.029983        0
        4362    945     94512   0.009999        0
        6448    907     906548  0.001000        2
        8612    925     9210084 0.000100        21
        8617    926     9260002 0.000100        21
Archivo:        C:\VBox-DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
Umb Disimilit.: 0.05
T. de Corte:    0.0001000
Max Individuos: 5000000000
        1930    984     7637    0.128846        0
        3559    1104    36829   0.029976        0
        4828    1126    112596  0.010000        0
        7493    1123    1122440 0.001000        2
        10163   1147    11412937        0.000100        25
        10167   1147    11470002        0.000100        25
Archivo:        C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
Umb Disimilit.: 0.03
T. de Corte:    0.0001000
Max Individuos: 5000000000
        4245    2716    8360    0.324880        0
        13550   3784    126133  0.030000        1
        18163   3992    399209  0.010000        1
        26729   3816    3814094 0.001000        7
        35727   3913    38935325        0.000100        72
        35742   3908    39080002        0.000100        73
Archivo:        C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
Umb Disimilit.: 0.03
T. de Corte:    0.0001000
Max Individuos: 5000000000
        4245    2716    8360    0.324880        0
        14672   4173    139099  0.030000        0
        19523   4298    429780  0.010000        0
        29565   4338    4335834 0.001000        7
        38918   4033    40129355        0.000100        72
        38921   4024    40240002        0.000100        72
Archivo:        C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
Umb Disimilit.: 0.03
T. de Corte:    0.0001000
Max Individuos: 5000000000
        4245    2716    8360    0.324880        0
        14194   4049    134967  0.030000        0
        18742   4120    411981  0.010000        1
        28865   4399    4396803 0.001000        7
        39236   4540    45177004        0.000100        80
        39282   4552    45529628        0.000100        81
Archivo:        C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
Umb Disimilit.: 0.03
T. de Corte:    0.0001000
Max Individuos: 5000000000
        4245    2716    8360    0.324880        0
        13775   3867    128899  0.030000        0
        17773   3789    378972  0.009998        1
        26704   3820    3818122 0.001000        7
        35789   3946    39263723        0.000100        69
        35818   3951    39510002        0.000100        69
Archivo:        C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
Umb Disimilit.: 0.03
T. de Corte:    0.0001000
Max Individuos: 5000000000
        4245    2716    8360    0.324880        0
        13303   3743    124777  0.029998        0
```

.\tmp\Simulacion001.csv

```
       17493   3751    375085  0.010000       1
       26313   3754    3752324 0.001000       6
       35095   3847    38280343       0.000100       69
       35107   3840    38400002       0.000100       69
Archivo:        C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
Umb Disimilit.: 0.01
T. de Corte:    0.0001000
Max Individuos: 5000000000
       5360    3918    8360    0.468660       0
       25644   7085    236164  0.030000       0
       33473   7136    713566  0.010000       1
       50050   7193    7189407 0.001000       10
       66875   7323    72867144       0.000100       109
       66928   7331    73312596       0.000100       110
Archivo:        C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
Umb Disimilit.: 0.01
T. de Corte:    0.0001000
Max Individuos: 5000000000
       5360    3918    8360    0.468660       0
       24984   6986    232890  0.029997       0
       32406   6899    689867  0.010000       1
       47356   6636    6633166 0.001000       9
       62516   6547    65144280       0.000100       95
       62549   6547    65473146       0.000100       96
Archivo:        C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
Umb Disimilit.: 0.01
T. de Corte:    0.0001000
Max Individuos: 5000000000
       5360    3918    8360    0.468660       0
       25028   7014    233798  0.030000       0
       32690   7088    708766  0.010000       1
       49411   7186    7182410 0.001000       9
       65900   7130    70945275       0.000100       104
       65920   7119    71190002       0.000100       105
Archivo:        C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
Umb Disimilit.: 0.01
T. de Corte:    0.0001000
Max Individuos: 5000000000
       5360    3918    8360    0.468660       0
       25020   6998    233264  0.030000       0
       32634   6988    698867  0.009999       0
       49393   7117    7113726 0.001000       10
       66150   7205    71691544       0.000100       102
       66173   7196    71965154       0.000100       102
Archivo:        C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
Umb Disimilit.: 0.01
T. de Corte:    0.0001000
Max Individuos: 5000000000
       5360    3918    8360    0.468660       0
       25600   7101    236698  0.030000       0
       33233   7084    708366  0.010000       1
       49707   7151    7147428 0.001000       10
       66082   7069    70338310       0.000100       104
       66099   7060    70600002       0.000100       104
Archivo:        C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
Umb Disimilit.: 0.04
T. de Corte:    0.0001000
Max Individuos: 5000000000
       3893    2369    8360    0.283373       0
       10838   3038    101297  0.029991       0
       14476   3157    315691  0.010000       1
       21412   3028    3027395 0.001000       6
       28280   2980    29661123       0.000100       56
       28285   2975    29750002       0.000100       56
Archivo:        C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
Umb Disimilit.: 0.04
T. de Corte:    0.0001000
Max Individuos: 5000000000
       3893    2369    8360    0.283373       0
       11775   3425    114186  0.029995       0
       15153   3281    328093  0.010000       1
       22677   3246    3244379 0.001000       6
       30297   3274    32577702       0.000100       61
       30319   3275    32750002       0.000100       62
Archivo:        C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
```

.\tmp\Simulacion001.csv

```
Umb Disimilit.: 0.04
T. de Corte:    0.0001000
Max Individuos: 5000000000
        3893    2369    8360    0.283373        0
        11515   3236    107866  0.030000        0
        15306   3397    339704  0.010000        0
        23170   3358    3356462 0.001000        6
        30463   3255    32388061        0.000100        63
        30487   3258    32581658        0.000100        63
Archivo:        C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
Umb Disimilit.: 0.04
T. de Corte:    0.0001000
Max Individuos: 5000000000
        3893    2369    8360    0.283373        0
        11730   3412    113733  0.030000        0
        15440   3375    337546  0.009999        1
        23608   3473    3471422 0.001000        7
        31520   3432    34155892        0.000100        64
        31528   3424    34247684        0.000100        65
Archivo:        C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
Umb Disimilit.: 0.04
T. de Corte:    0.0001000
Max Individuos: 5000000000
        3893    2369    8360    0.283373        0
        11352   3181    106033  0.030000        0
        14865   3186    318647  0.009999        0
        22060   3139    3137433 0.001000        5
        29467   3252    32363516        0.000100        58
        29478   3248    32480445        0.000100        58
Archivo:        C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
Umb Disimilit.: 0.04
T. de Corte:    0.0001000
Max Individuos: 5000000000
        3893    2369    8360    0.283373        0
        10922   3037    101250  0.029995        0
        14226   3046    304586  0.010000        1
        20924   2961    2959754 0.001000        6
        28486   3187    31714784        0.000100        60
        28496   3185    31853156        0.000100        61
Archivo:        C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
Umb Disimilit.: 0.04
T. de Corte:    0.0001000
Max Individuos: 5000000000
        3893    2369    8360    0.283373        0
        11412   3264    108808  0.029998        0
        15310   3427    342684  0.010000        0
        23342   3408    3406448 0.001000        6
        31571   3562    35442788        0.000100        67
        31583   3555    35550983        0.000100        67
Archivo:        C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
Umb Disimilit.: 0.04
T. de Corte:    0.0001000
Max Individuos: 5000000000
        3893    2369    8360    0.283373        0
        11449   3314    110477  0.029997        0
        15382   3484    348384  0.010000        1
        23393   3550    3548227 0.001000        7
        31296   3446    34291464        0.000100        68
        31316   3449    34491520        0.000100        68
Archivo:        C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
Umb Disimilit.: 0.04
T. de Corte:    0.0001000
Max Individuos: 5000000000
        3893    2369    8360    0.283373        0
        10376   2871    95700   0.030000        1
        13268   2767    276688  0.010000        1
        19357   2650    2648677 0.001000        5
        25064   2502    24895524        0.000100        48
        25075   2501    25010735        0.000100        49
Archivo:        C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
Umb Disimilit.: 0.04
T. de Corte:    0.0001000
Max Individuos: 5000000000
        3893    2369    8360    0.283373        0
        10670   2976    99200   0.030000        0
```

.\tmp\Simulacion001.csv

14189	3051	305100	0.010000	0
21662	3198	3196403	0.001000	5
29466	3374	33572141	0.000100	63
29475	3369	33692792	0.000100	64

Además se generarán otros archivos temporarios como 'Tempddd.txt', los que se irán eliminando a medida que se ejecute el programa 'leer-List.exe'.

Al terminar de ejecutarse todos los hilos de ejecución de 'leerList.exe', el programa interactivo 'LeerList.jar' ejecutará el programa 'genearinforme.exe' que tomará todos los archivos temporarios 'Simulacionddd.csv' generando dos únicos archivos de salida 'Simulacion.csv' y 'SimulacionR.csv' . Este ultimo se muestra en la Figura 8.

Figura 8

SimulacionR.csv - Volver

```
Archivo:        C:\VBox-DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
Umb Disimilit.: 0.02
T. de Corte:    0.0001000
Max Individuos: 5000000000
Corridas: 10 - Cant. Estimada de especies: 21622 - Tiempo total de ejecución: 416

Archivo:        C:\VBox-DiscoCompartido\DatosC\err023721.good.filter.fasta.phy.fn.list
Umb Disimilit.: 0.05
T. de Corte:    0.0001000
Max Individuos: 5000000000
Corridas: 10 - Cant. Estimada de especies: 10085 - Tiempo total de ejecución: 257

Archivo:        C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
Umb Disimilit.: 0.01
T. de Corte:    0.0001000
Max Individuos: 5000000000
Corridas: 10 - Cant. Estimada de especies: 66583 - Tiempo total de ejecución: 1063

Archivo:        C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
Umb Disimilit.: 0.03
T. de Corte:    0.0001000
Max Individuos: 5000000000
Corridas: 10 - Cant. Estimada de especies: 36969 - Tiempo total de ejecución: 729

Archivo:        C:\VBox-DiscoCompartido\DatosC\S86.phy.cgi.fn.list
Umb Disimilit.: 0.04
T. de Corte:    0.0001000
Max Individuos: 5000000000
Corridas: 20 - Cant. Estimada de especies: 29528 - Tiempo total de ejecución: 1212
```

Observación: cuando se manden a 'Procesar' las simulaciones seleccionadas en 'LeerList.jar', este dejará de ser visible mostrando un diálogo , hasta que se haya concluído la simulación en que mostrará otro diálogo y el programa interactivo volverá a estar visible. La pantalla del programa 'LeerList.jar' podrá ser redimensionada a voluntad, y las columnas de los archivos seleccionados podrán ser reubicadas a voluntad o necesidad del usuario. Se podrán seleccionar hasta un máximo de 29 (veintinueve) simulaciones, lo que ex-cede ampliamente las necesidades, y desde 1 (uno) a 99 (noventa y nueve) 'Corridas a Ejecutar Sobre Cada Muestra' lo mismo que para la cantidad de 'Hilos de Ejecución' Hay que tener en cuenta que este programa ejecuta dos programas escritos en Lenguaje C: 'leerList.exe' y luego 'genearinforme.exe' Se hace notar que se ha elegido el lenguaje Java (compile una vez y ejecute en cualquier plataforma), para el programa interactivo, En tanto que

para los programas en Lenguaje C se ha elegido el entorno Code::Blocks con compilador MingW (compilador gcc, port de Unix/Linux al entorno Windows), y que con este compilador casi se garantiza el 'codifique una vez y compile y ejecute en cualquier plataforma'. Antes que el programa interactivo 'LeerList.jar' invoque al 'leerList.exe', genera tantos archivos de texto como 'Hilos de Ejecución' se hayan seleccionados. En el caso de esta corrida se generan los siguientes dos archivos de texto con los tres primeros cam-pos de longitud fija y el tercero de longitud variable. Como se puede ver los tres primeros corresponden al 'Umbral de Disimilitud', el 'T de corte' y la 'Cantidad de Indivi-duos a Generar' siendo el último el nombre (con path completo), del archivo a procesar. El programa 'leerList.exe' será invocado con el nombre del archivo de texto que contiene los parámetros necesarios para cumplir con su cometido, es decir que se ejecutarán (siguiendo este ejemplo) los comandos: '.\leerList.exe .\tmp\LeerList001.txt' y '.\leerList.exe .\tmp\LeerList002.txt', cada uno en un hilo de ejecución separado.
Una vez hecho, el programa interactivo 'LeerList.jar' quedará a la espera que terminen todos los hilos de ejecución que ha pedido, para lanzar un proceso final '.\genearinforme.exe 002' (en este caso).

Cada hilo de ejecución del programa 'leerList.exe' al comenzar a ejecutarse reconoce el argumento recibido (nombre del archivo de texto), y por cada línea de texto del mismo, abre el archivo indicado en el último campo, selecciona el 'Umbral de Disimili-tud' del primer campo y genera un archivo temporario v.g.: '.\tmp\Temp001.txt', '.\tmp\Temp002.txt', etc. (si hubiera más de dos hilos de ejecución), para el umbral indicado, procediendo a generar la simulación que terminará al alcanzar el 'T de corte' o la 'Cantidad de Individuos a Generar' (lo que suceda primero), generando en su archivo '.\tmp\Simulacion001.csv', o '.\tmp\Simulacion002.csv', (según qué hilo de ejecución sea). Luego, cuando el programa 'LeerList.jar' ejecute el comando '.\genearinforme.exe 002' (como en este caso), el programa recibe la cantidad (002) de archivos '.\tmp\Simulacionddd.csv' a procesar. Su misión es leer estos archivos temporarios de la simulación, ordenarlos de menor a mayor por nombre de archivo + umbral, y generar el archivo con la simulación detallada: '.\Simulacion-.csv', y el archivo con la simulación resumida: '.\SimulacionR.csv'.
Observación: Se ha elegido la generación de las salidas '.\Simulacion-.csv' y '.\SimulacionR.csv' con un formato de campo separado por un carácter de Tabulación (una de las versiones simplificadas de los archivos CSV comma/character separated value dice: "A comma-separated values (CSV) (also sometimes called character-separated values, because the separator character does not have to be a comma) file stores tabular data (numbers and text) in plain-text form. Plain text means that the file is a sequence of characters, with no data that has to be interpreted as binary numbers. A CSV file consists of any number of records, separated by line breaks of some kind; each record consists of fields, separated by some other character or string, most commonly a literal comma or tab. Usually, all records have an identical sequence of fields.A general standard for the CSV file format does not exist, but RFC 4180 provides a de facto standard for some aspects of it."). De este modo estos archivos pueden ser abiertos con una planilla de cálculo, para su formateo y visualización.

En el Anexo 1 se adjuntan los códigos de los programas elaborados.

5. RESULTADOS

5.1- Para la población simulada construida con la distribución de Fischer de parámetros $\alpha = 5000$ y x=0.995 la prueba experimental consistió en aplicar el algoritmo ARE, sobre una muestra inicial de 1000 individuos, realizando distintas cantidades de iteraciones. En cada corrida se obtuvo también el valor que alcanzó la probabilidad de especie nueva T_i al cortar la simulación. La Tabla 2 exhibe los resultados.

Tabla 2

Estimación ARE de la riqueza comunitaria según cantidad de iteraciones

ARE/60000 Iteraciones	ARE/200000 Iteraciones	ARE/500000 Iteraciones	ARE/897541 Iteraciones
14615	19271	24559	25327
$T_{60000} = 0.091$	$T_{200000} = 0.026$	$T_{500000} = 0.011$	$T_{897541} = 0.005$

En base a la misma muestra de 1000 individuos simulados se comparó el desempeño de los estimadores no paramétricos CHAO y ACE con las estimaciones ARE obtenidas para cantidades crecientes de iteraciones. A su vez todos estos resultados se compararon con la cantidad real de especies. La Tabla 3 evidencia las mejoras que ARE produce en la estimación de riqueza respecto de CHAO y ACE conforme aumenta el número de iteraciones.

Tabla 3

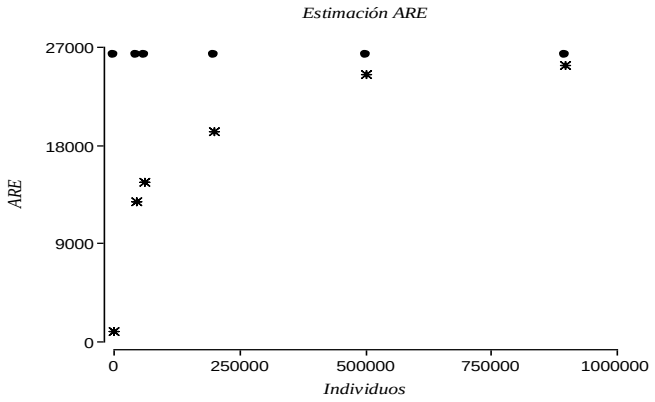
Comparación del desempeño de estimadores de riqueza

Real	CHAO	ACE	ARE/45000	ARE/60000	ARE/200000	ARE/500000
26332	6699	6751	12821	14615	19271	24559
100%	25%	26%	49%	56%	73%	93%

La Figura 9 muestra como las estimaciones ARE van aproximando la cantidad real de especies presentes en la población según crece el número de iteraciones realizadas. Se observa que la serie obtenida va buscando un valor asíntótico que no supera a S.

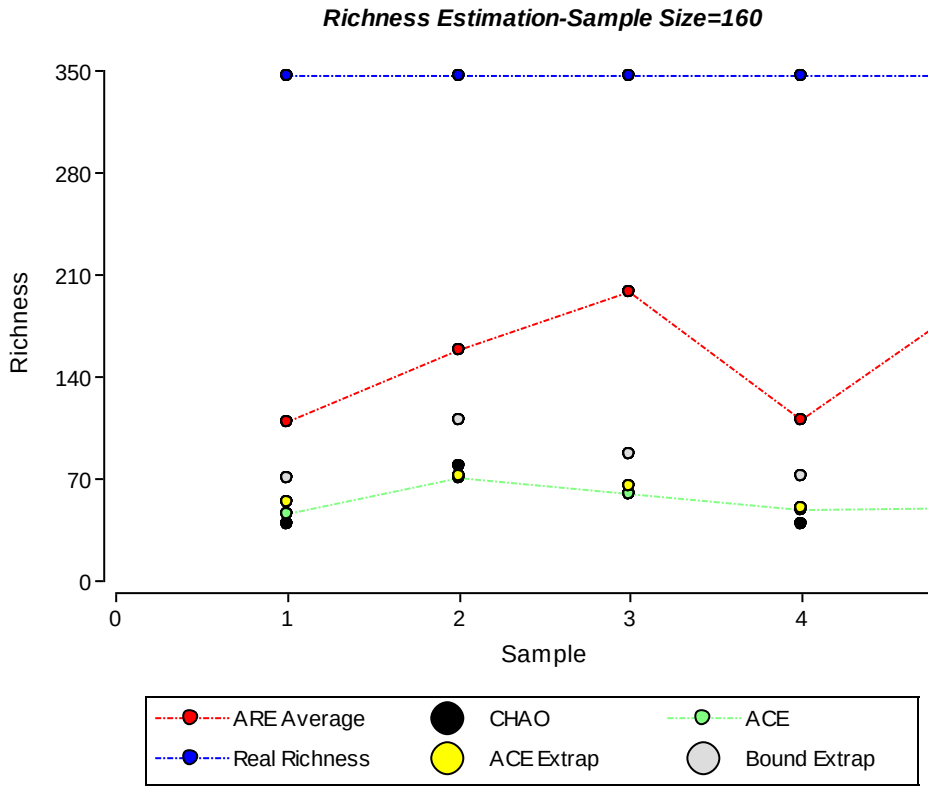
Figura 9

Aproximación de la Cantidad de Especies por ARE



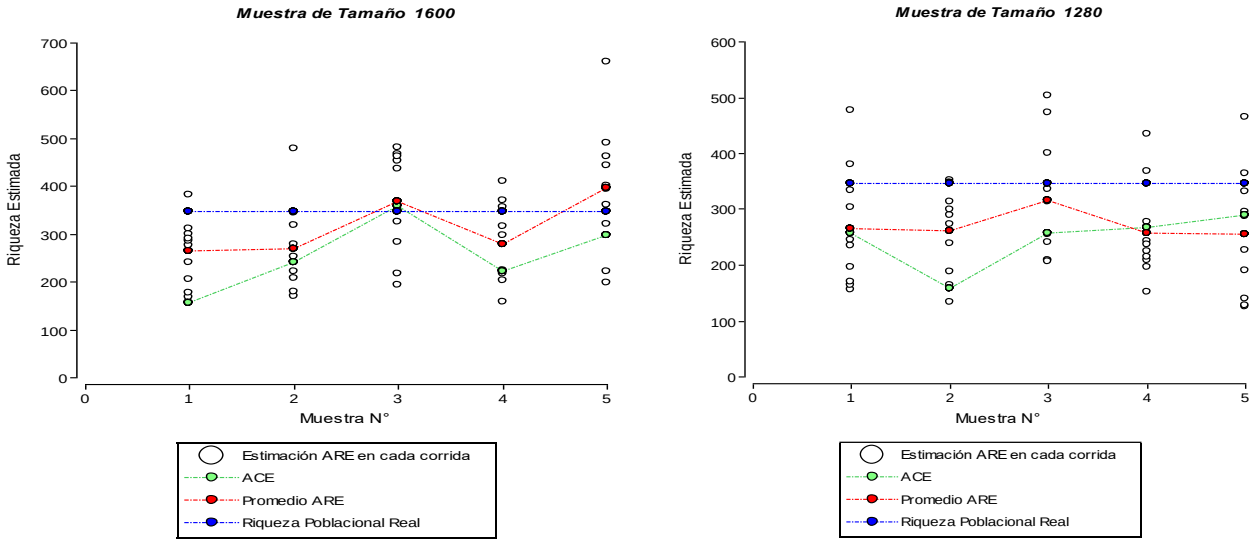
Al utilizar el conjunto FS396,archea para probar el desempeño de ARE se estudió primero la estimación basada en 5 muestras de 160 individuos que representa el 1% aproximadamente del total de los 16316 individuos que conforman la comunidad. En cada caso se realizaron 10 corridas del algoritmo comparando el promedio de las estimaciones ARE con las estimaciones CHAO y ACE, con la extrapolación y su cota, y con el valor real de la riqueza $S = 346$. Los resultados se exhiben en la Figura 10.

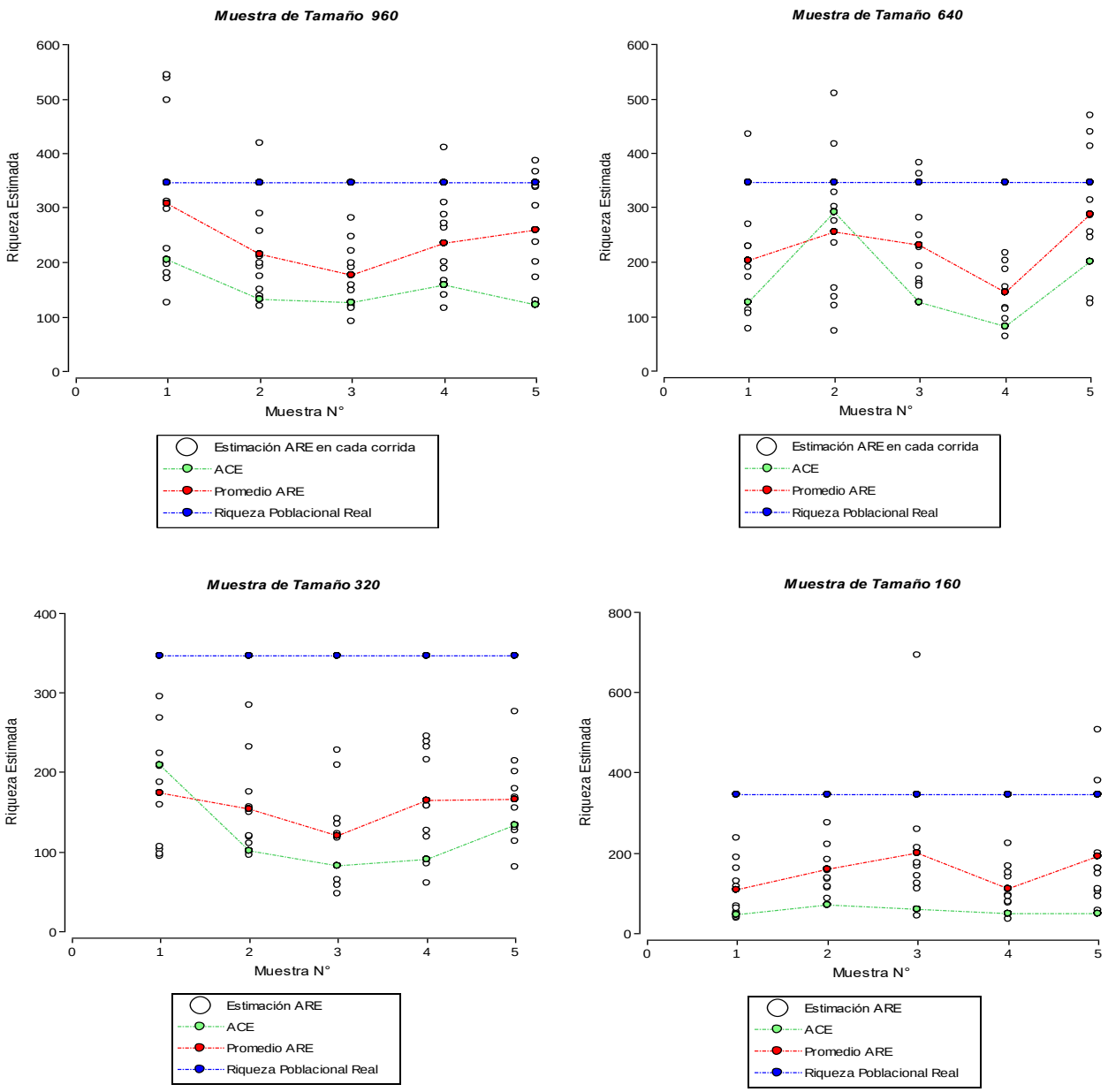
Figura 10



A continuación se tomaron 5 muestras de cada uno de los tamaños 320, 640, 960, 1280 y 1600. Sobre cada muestra se realizaron 10 corridas del algoritmo estableciéndose como estimación el promedio de las estimaciones para cada corrida. La intención fue comparar las estimaciones ACE y ARE con la riqueza real conocida. La Figura 11 muestra los resultados obtenidos en cada corrida para cada muestra y tamaño muestral.

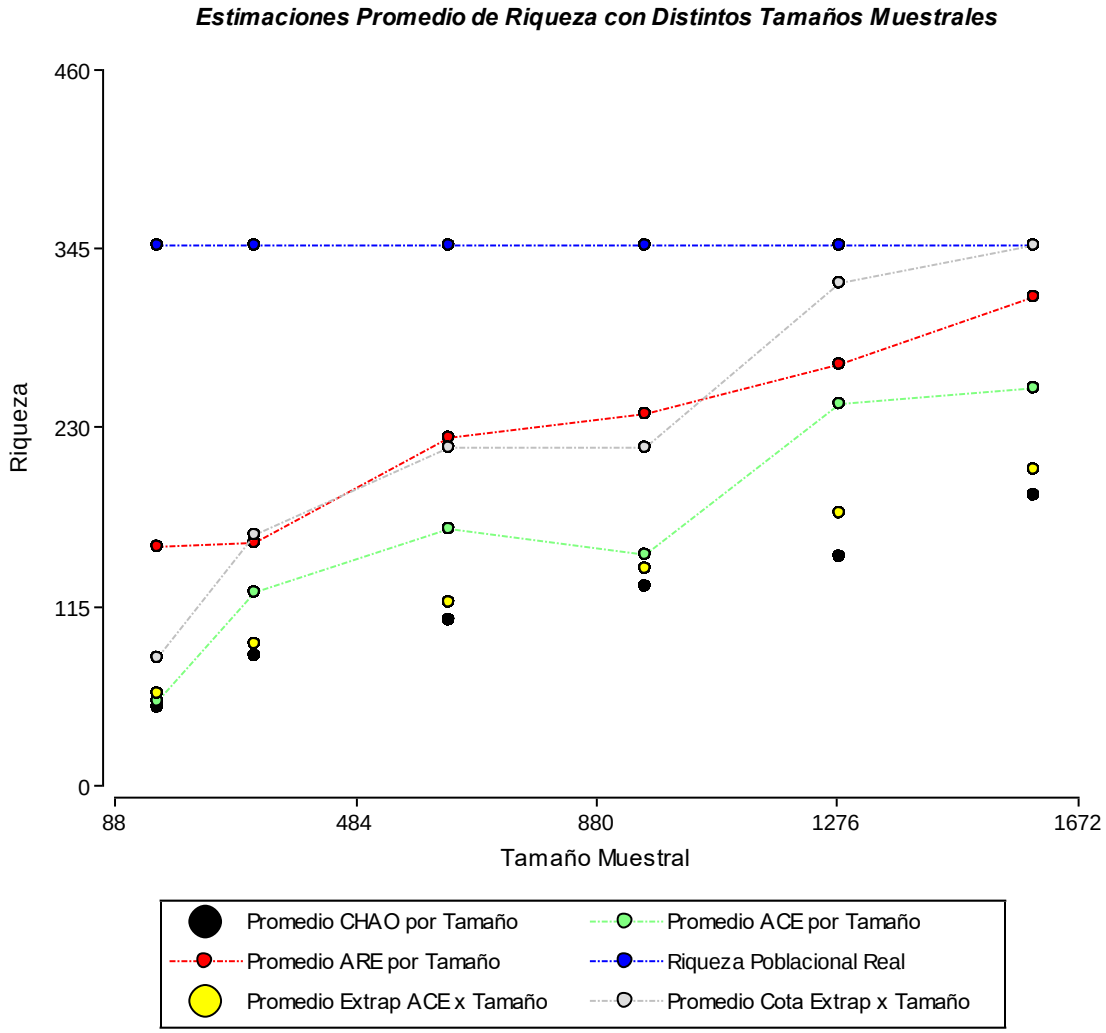
Figura 11





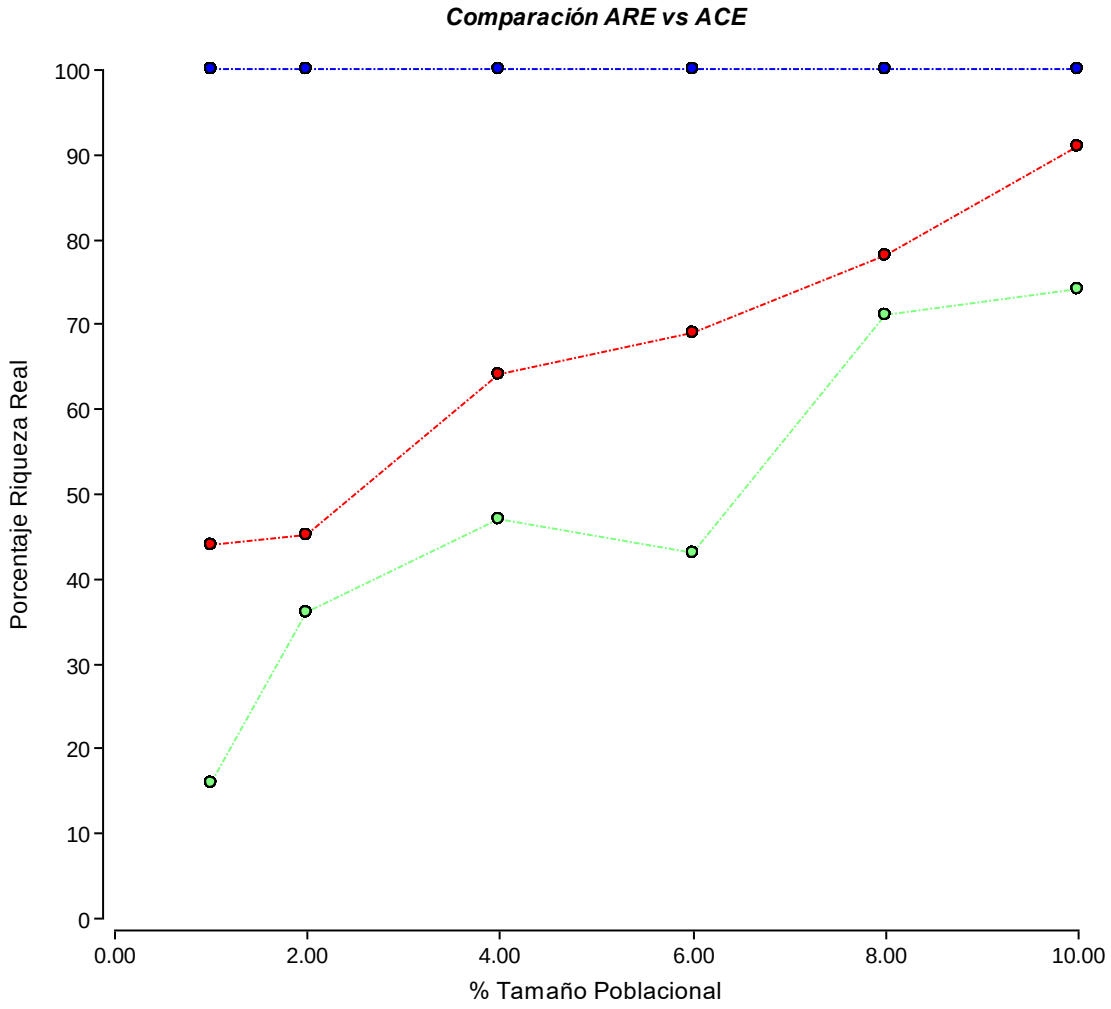
Aunque en la práctica ecológica pueda escogerse una sola muestra de la comunidad, para cada tamaño muestral se promediaron cada uno de los estimadores a efecto de compararlos en términos estadísticos. Así pudo verse que el aumento del tamaño muestral produce, como era esperable, un mejor desempeño promedio de todos ellos como se ve en la Figura 12.

Figura 12



En particular, cuando se observan los resultados en porcentual de la riqueza obtenidos por ARE y ACE según el tamaño porcentual de la muestra con referencia a la población se aprecia que ARE mejora una en la estimación como se muestra en la Figura 13

Figura 13



Las diferencias de estimación porcentuales a favor de ARE sobre el 100% de la riqueza resultan, para el caso analizado, las de la Tabla 4.

Tabla 4

Mejora Porcentual de la Estimación ARE

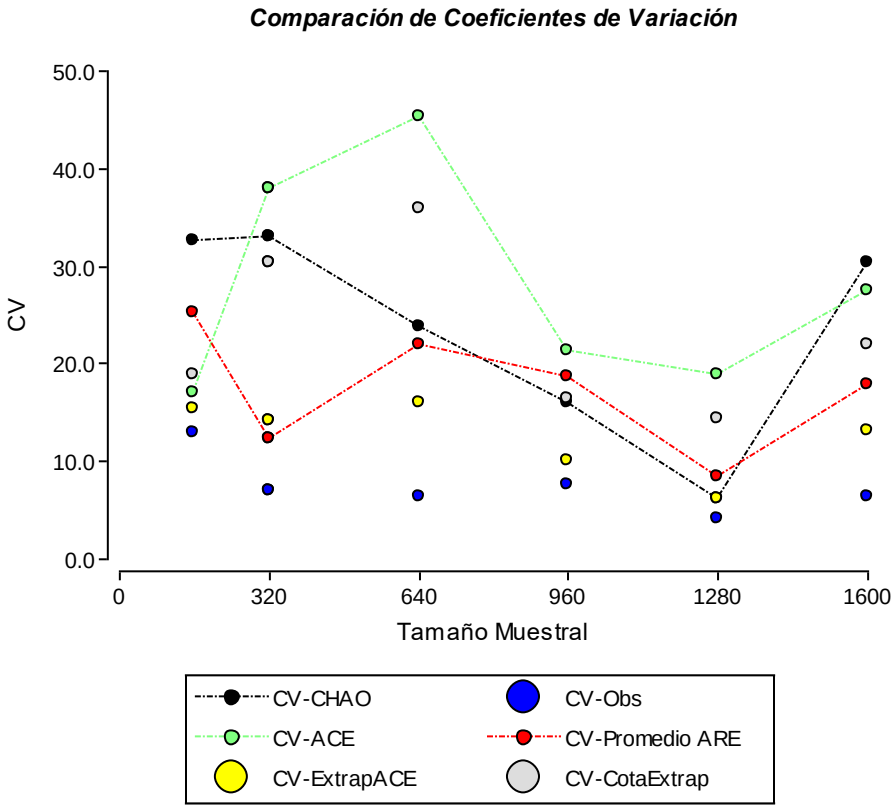
Porcentaje Tamaño Muestral	1%	2%	4%	6%	8%	10%
Mejora ARE	28%	21%	33%	32%	7%	17%

Se observa que aún para tamaños muestrales muy grandes porcentualmente, la estimación ARE produce una mejora respecto del estadístico no paramétrico ACE.

Se efectuó además un estudio de la variabilidad midiendo el coeficiente de variación para establecer la estabilidad de cada uno de los estimadores obtenidos de las 5 muestras de cada tamaño muestral. Como estimación de ARE se tomó el promedio obtenido en base a las 10 corridas efectuadas para cada muestra. Se consideraron los valores de CHAO, ACE, Extrapolación y Cota de Extrapolación obtenidos de cada muestra, y como valores observados se tomaron las cantidades de especies obtenidas en cada una de las 5 muestras para cada tamaño. Estos últimos

valores mostrarán entonces la variabilidad de la riqueza muestral. Los resultados se exhiben en la Figura 14.

Figura 14



Un análisis pormenorizado de los resultados expuestos se realiza en la siguiente sección.

5.2- Se realizó una prueba inicial de funcionamiento del nuevo programa utilizando como datos el mencionado archivo S86.phy.cgi.fn.list para los umbrales de disimilitud 0.01, 0.03 y 0.04 y valor T de corte en 0.0001. También se probó con el archivo err023721.good.filter.fasta.phy.fn.list correspondiente a suelo de la Selva Amazónica, en Brasil, con coordenadas son (3.433333 S, 60.383333 O) salido del procesamiento con MOTHUR. En este caso los umbrales de disimilitid utilizados fueron 0.02 y 0.05. Las estimaciones respectivas de la cantidad de especies se exhiben en la Figura 16 y resultan coherentes con las obtenidas por la version en código R del algoritmo realizadas individualmente durante la ejecución del proyecto de investigación anterior C112. El tiempo de ejecución se redujo en promedio a la décima parte.

6. DISCUSIÓN

6.1- El uso de la distribución de Fischer para construir una población simulada se basa en que se ha observado, en particular en ciertas biosferas marinas, que los filotipos estadísticamente raros se ajustan a ella [26] (Galand et al. 2009). La simulación de la población en base a tal distribución requiere elegir los parámetros α y x que intervienen en el cálculo. El número máximo de individuos que aparecen en al menos un cluster crece en la medida en que crece el tamaño poblacional y potencialmente, si se supusiera una población infinita, podría aceptarse que $m \to \infty$.

En tal caso la cantidad de especies $S = \sum_{k=1}^{\infty} \frac{\alpha}{k} x^k = -\alpha \ln(1-x)$ y la cantidad total de individuos $N = \sum_{k=1}^{\infty} k \frac{\alpha x^k}{k} = \frac{\alpha x}{1-x}$. Para deducir ambas fórmulas se ha tenido en cuenta que $0 \prec x \prec 1$ (Fischer et al. 1943). Por otra parte se analiza que la cantidad α es un parámetro intrínseco de la riqueza de cada población y puede utilizarse como índice de diversidad (Magurran 2004). Si se establecen de antemano valores para la cantidad de especies S y el tamaño N de la población puede determinarse el valor de x al resolver, por procedimientos iterativos, la ecuación $\frac{S}{N} = \left[(1-x)/x\right]\left[-\ln(1-x)\right]$ que expresa la razón entre la cantidad de especies y el tamaño poblacional. El parámetro x depende entonces de esta razón y en la práctica se cumple $x \succ 0.9$ sin que, por supuesto, pueda superar el valor 1 (Magurran 2004). La Tabla 5 proporciona los valores de la razón $\frac{S}{N}$ calculados con distintos valores de x al suponer que el tamaño poblacional es $N = 1000000$. De allí es posible obtener el número de especies S y con él, el valor del parámetro α que puede calcularse a partir de N y x según $\alpha = \frac{N(1-x)}{x}$. Se observa que, en todos los casos resulta $\alpha \prec S$. Como αx es el primer término de la serie log, si x está cercano a 1, resulta aproximadamente el número esperado de especies que aparecen una vez en la población, es decir la mínima cantidad de especies que pueden considerarse raras. Además cuando decrece la razón $\frac{S}{N}$ entre la cantidad de especies y el tamaño poblacional, que en algún sentido expresa riqueza, α también lo hace. En la práctica el índice de riqueza α solo puede calcularse a partir de muestras y en tal caso su valor puede considerarse independiente del tamaño de la muestra cuando esta tiene mas de 1000 individuos. (Magurran 2004)

Tabla 5

Relaciones entre parámetros

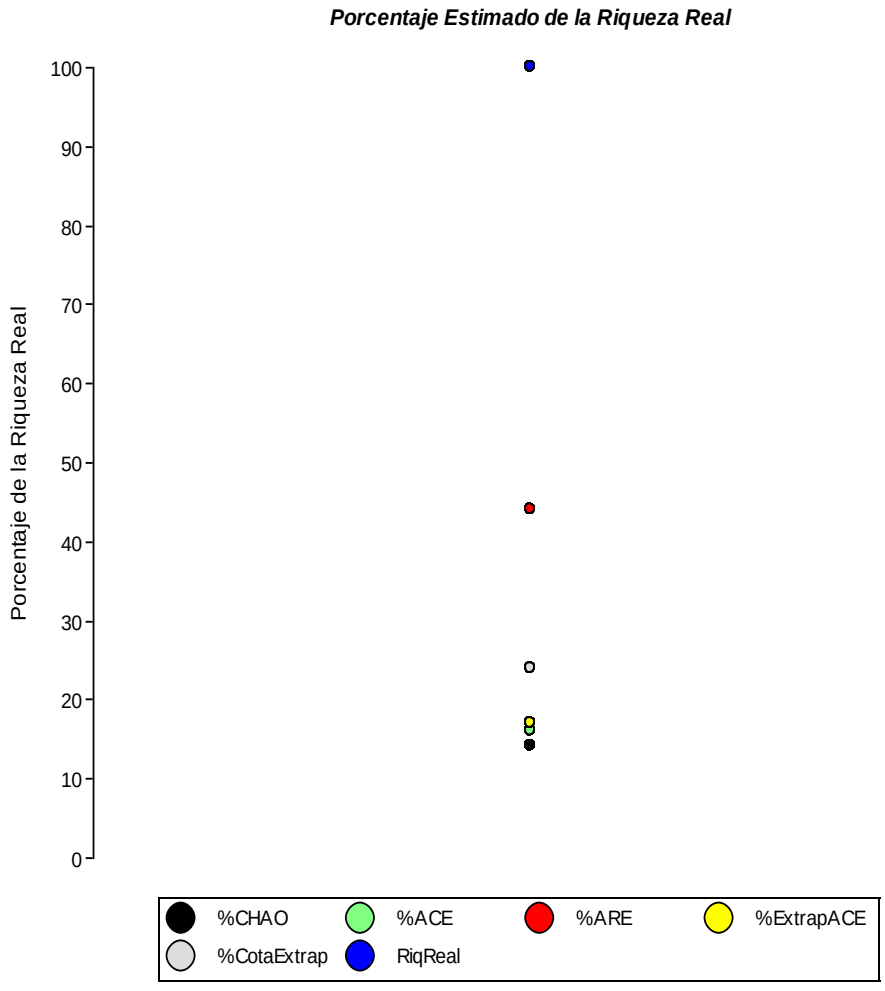
x	S/N	S	α
0.9	0.2558 ≈ 1/4	255817	111111.11
0.95	0.1577 ≈ 1/6	157700	52631.58
0.975	0.0946 ≈ 1/11	94600	25641.03
0.99	0.0465 ≈ 1/22	46500	10101.01
0.995	0.0266 ≈ 1/38	26600	5025.13
0.9975	0.0150 ≈ 1/67	15000	2506.27
0.999	0.0069 ≈ 1/145	6900	1001

Si se considera una aproximación inicial de la riqueza entre 5000 y 30000 especies para una población de 1 millón de individuos los valores correspondientes *x* y α deberán oscilar entre $0.995 \le x \le 0.999$ y $1000 \le \alpha \le 5000$ respectivamente. Para los parámetros elegidos *α= 5000* y x=0.995 el valor $\alpha x = 5000 \times 0.995 = 4975$ representa aproximadamente la cantidad de especies de mayor rareza de la comunidad. Estas especies, representan al 19% del total. A su vez según la distribución simulada habría un 10% de especies representadas por dos individuos, un 6% con tres individuos en la población y así. En suma; hay una importante proporción de especies raras en la comunidad.

Cuando se seleccionó una muestra de 1000 individuos sobre un total comunitario 897541 los resultados volcados en la Tabla 2 permitieron comprobar la creciente eficacia de la estimación ARE conforme aumentaba el número de individuos simuladamente agregados a la muestra. También aquí pudo verse la disminución hacia 0 de la estimación de Turing de especie nueva. La Tabla 3 muestra la proporción de la riqueza real detectada según aumenta la cantidad de iteraciones. Al llegar a las 500000 el porcentaje estimado por ARE es del 93% de la riqueza lo que resulta un desempeño cualitativamente diferente al aportado por ACE (26%) y CHAO (25%) a partir de la misma muestra.

Para seleccionar la muestra F396.archea y utilizarla como una población de prueba se tuvo en cuenta su tamaño, relativamente grande, y la forma asíntótica que considerados todos los individuos, presenta su curva de rarefacción (Haegeman et al. 2013). Al seleccionar 5 muestras de 160 individuos se observó que la estimación ARE era superior a la obtenida como cota de la extrapolación y prácticamente duplicaba los cálculos hechos por ACE, CHAO y la propia extrapolación cuando se agregaba una cantidad de individuos ideales que triplicaba el tamaño muestral. $m = 3n$ es el valor máximo recomendado, adoptando un punto de vista estadístico, para realizar la extrapolación (Chao et al. 2014). Sin embargo la experiencia permitió también constatar que la mejor estimación proporcionada por ARE apenas alcanzaba a la mitad de la riqueza real. En tal sentido se promedió cada estimación considerando las 5 muestras para obtener una gráfica que ilustra sobre el porcentaje de la riqueza real estimado por cada estadístico. En la Figura 15 se pueden apreciar las diferencias significativas entre las estimaciones promedio y el valor real.

Figura 15



El incremento del tamaño muestral permitió acortar estas diferencias. En la Figura 11 se observa que para las muestras de 1600 individuos, alrededor del 10% del tamaño poblacional, se dieron casos de estimación muy precisa y aún de sobre estimación. En términos generales las Figura 12 y 13 confirman el mejor desempeño de ARE (línea roja) respecto del estimador no paramétrico ACE (línea verde). También se observa un comportamiento de la extrapolación con $m = 3n$ parecido al de CHAO produciendo ambos subestimación de la riqueza. La cota de extrapolación está más cercana a los valores obtenidos por ARE, especialmente cuando el tamaño de la muestra crece. Sin embargo, para el tamaño muestral más pequeño ARE proporciona una estimación mejor. Como era de esperar, además, una estimación más precisa se da cuando aumenta el tamaño de la muestra original.

No obstante lo analizado, los resultados no dejan de evidenciar las diferencias en la precisión de la estimación obtenida en el caso simulado por la distribución de Fischer y el caso real analizado, que sugieren el estudio de otras medidas para dar cuenta de la diversidad comunitaria, sobretodo en el caso en que los análisis deban realizarse a partir de una sola muestra.

En la Figura 14 se analiza la variabilidad de las distintas estimaciones. En primer lugar hay que mencionar la variabilidad propia del muestreo. Ésta puede apreciarse a través de la variación de la cantidad de especies distintas que se presentan en diferentes las muestras. Son los puntos coloreados en azul y exhibe la menor variación relativa para todos los tamaños muestrales. De acuerdo a esto, el incremento de la variación relativa para cada tipo de estimación estará ligado a la segunda fuente de variabilidad que es la inherente al método aplicado para establecerla. Las estimaciones no paramétricas revelan porcentajes de variación entre 6 % y 33 % para CHAO y entre 17% y 45% para ACE, con rangos respectivos de 27% y 28% de variación. ARE revela menores porcentajes, entre 8% y 25%, y a su vez menor rango de variación del mismo, 17%. De tal forma la estimación ARE aprecia mejor la riqueza y lo hace con menor variabilidad.

Nota: se adjuntan artículos producidos y presentaciones realizadas en congresos

7. CONCLUSIONES

- Se ha presentado un enfoque alternativo basado en un modelo experimental que “ajusta” al proceso de ampliación de la muestra necesario para hacer más precisa la estimación de la riqueza. Tal enfoque, emparentado con la minería de datos, utiliza un subconjunto inicial para predecir el valor de un parámetro de la comunidad.
- Los resultados obtenidos han mejorado sensiblemente las estimaciones de riqueza realizadas en base a modelos y supuestos de tipo analítico - estadísticos.
- De todas formas ha vuelto a quedar en claro la necesidad de evaluar la biodiversidad a través de distintas medidas que completen el conocimiento sobre la comunidad y proporcionen una apreciación más fidedigna de sus propiedades.
- Habida cuenta del mejor, aunque todavía insuficiente, desempeño alcanzado en la estimación de la riqueza de acuerdo al enfoque alternativo propuesto, queda sugerida su aplicación a otras medidas que cuantifiquen la cantidad de información y la distribución de especies en la comunidad.
- Se ha elaborado un programa con características interactivas que permite la ejecución en paralelo del algoritmo sobre distintos archivos,

reiteradamente, con diferentes umbrales de disimilaridad y diferentes valores de corte.

8. PRODUCCIÓN CIENTÍFICO-TECNOLÓGICA

8.1- Publicaciones

a) Artículos

- Santa María, Cristóbal R. y Soria, Marcelo A. “Evaluation of richness in microbial communities “. Presentado como Original Paper para evaluar su publicación en Bioinformatics. Oxford University Press el 22/01/2015

b) Congresos Internacionales, Nacionales, Simposios, Jornadas, otros

- Santa María, Cristóbal R. y Soria, Marcelo A. “Inferencia de Parámetros de Biodiversidad por Simulación” Ponencia. 4to Congreso Argentino de Matemática Aplicada, Computacional e Industrial. Buenos Aires. Argentina.Mayo de 2013.Asociación Argentina de Matemática Aplicada, Computacional e Industrial (ASAMACI).Artículo completo. Revista MACI Volúmen IV.5-8 ISSN 2314-3282.
- Santa María, Cristóbal R. y Soria, Marcelo A. “Simulation applied to the Estimation of Microbial Richness”. Póster. 4to Congreso Argentino de Bioinformática y Biología Computacional y IV Conferencia Internacional de la Sociedad Iberoamericana de Bioinformática. Rosario. Argentina. Octubre de 2013. Asociación Argentina de Bioinformática y Biología Computacional. (A2B2C) Artículo breve. Proceedings del Congreso.
- López, Luis; Martínez, Pablo; Cacho Mendoza, Ariel; Soria, Marcelo A. y Santa María, Cristóbal R. “Simulación en Evaluaciones de Biodiversidad”. Póster. WICC 2014 XVI Workshop de Investigadores en Ciencias de la Computación.. Ushuaia. Argentina. Mayo 2014. Red UNCI. Artículo completo. Ediciones WICC. ISBN 978-950-34-1084-4. Págs. 158-162

En el Anexo 2 se presentan los trabajos citados.

9. BIBLIOGRAFÍA

Armougom, F. and Raoult, D. 2009. Exploring Microbial Diversity Using 16S rRNA High-Throughput Methods. Journal of Computer Science & Systems Biology Vol-ume 2(1): 074-092 (2009) – 074

Chao, A. 1984. Nonparametric estimation of the number of classes in a population. Scand J Statist 11: 265-270.

Chao, A and Lee, S. 1992. Estimating the Number of Classes via Sample Coverage. Journal of American Statistical Association. Volume 87. Issue 417.

Chao, A and Shen, T. 2003 Nonparametric estimation of Shannon’s index of diversi-ty when there are unseen species in sample. Environmental and Ecological Statistics 10, 429-443.

Chao, A. Gotelli, N. J. Hsieh, T.C. Sander, E. L. Ma, K.H. Colwell, R. K. and Ellison A. M. 2014. Rarefaction and extrapolation with Hill numbers: a framework for sampling and estimation in species diversity studies. Ecological Monographs. 84 (1) 2014 pp.45-67.

Fischer, R. Corbett, S. and Williams, C. 1943. The Relation Between the Number of Species and the Number of Individuals in a Random Sample of an Animal Popula-tion. The Journal of Animal Ecology. British Ecological Society. Vol 12 N°1 (1943)

Galand, P. E. Casamayor, E. O. Kirchman, D. L. and Lovejoy, C. 2009. Ecology of the rare microbial biosphere of the Artic Ocean. PNAS. Vol. 106 no. 52 22427-22432.
Good, I. 1953. “The Population Frequencies of Species and Estimation of Popula-tion Parameters”. Biometrika. Vol 40 N° 3/4.
Gotelli, N and Colwell,R. 2001. Quantifying biodiversity: procedures and pitfalls in measurement and comparison of species richness. Ecology Letters. 4: 379-391
Haegeman, B. Hamelin, J. Motitarty, J. Neal, P. Dushoff, J. and Weitz, J. 2013 Robust estimation of microbial diversity in theory and in practice. ISME Journal 2013, 1-10
Hill, M. O. 1973. Diversity and Eveness: A Unifying Notation and Its Consequeces. Ecology. Vol.54. No 2 pp 427-432.
Hill, T. Walsh, K. Harris, J. and Moffett, B. 2003. Using Ecological Diversity Measures with Bacterials Communities. FEMS.Microbiology Ecology 43 1-11
Hillis, D. M. Moritz, C. and Mable, B. K. 1996. Molecular Systematics. Second Edition, Sinauer Associates, Inc. Publishers. Sunderland, MA. USA.
Hollister, E, Engledow, A, Hammett, A, Provin, T, Wilkinson, H. y Gentry, T. 2010. Shifts in microbial community structure along an ecological gradient of hypersaline soils and sediments. The ISME Journal. 1-10.
Huber, J. A. Mark Welch, D. B. Morrison, H. G. Huse, S. M. Neal, P.R. Butterfield, D. and A. Sogin, M. L. 2007. Microbial Population Structures in the Deep Marine Biosphera. Science 318, 97(2007) DOI: 10.1126/science.1146689
Hughes, J. Hellmann, J. Ricketts, T. and Bohannan, B. 2001. Counting the uncounta-ble: statistical approaches to estimating microbial diversity. Applied and Environ-mental Microbiology. 4399-4406.
Hughes,J and Hellman, J. 2005. The Application of Rarefaction Techniques to Molecular Inventories of Microbial Diversity. Methods in Enzymology. Vol 397.
Huse, S. M. Welch, D.M. Morrison, H.G. and Sogin, M.L. 2010. Ironing out the wrinkles in the rare biosphere through improved OTU clustering Environmental Microbiology (2010) 12(7), 1889–1898
Magurran, A. 2004. Measuring Biological Diversity. Blackwell Science Ltd.
Magurran, A and McGill, B.J. 2011. Biological Diversity. Oxford University Press.
Nádas, A. 1985. On Turing’s Formula for Word Probabilities. IEEE Transactions on Acoustics, Speech and Signal Processing. Vol ASSP-33 N° 6.
O’Hara, R. 2005. Species richness estimators: how many species can dance on de head of a pin. Journal of Animal Ecology. 74, 375.386
Roesch, L. Fulthorpe, R. Riva, A. Casella, G. Hadwin, A. Kent, A. Daroub, S. Ca-margo, F. Farmerie, W. and Triplett, E. 2007. Pyrosequencing enumerates and con-trasts soil microbial diversity. The ISME Journal. 1, 283-290.
Schloss, P and Handelsman, J. 2005. Introducing DOTUR, a Computer Program for Defining Operational Taxonomic Units and Estimating Species Richness. Applies and Environmental Microbiology. pp 1501-1506
Schloss, P. and Handelsman,J. 2006. Toward a census of bacteria in soil. PLoS Computational Biology. Volume 2.
Schloss, P. 2010. The Effects of Alignment Quality, Distance Calculation Method, Sequence Filtering, and Region on the Analysis of 16S rRNA Gene-based Studies. PLoS Computational Biology 6(7): e1000844. doi:10.1371/Journal.pcbi.1000844.
Youssef, N. and Elshahed, M. 2008. Species richness in soil bacterial communities: A proposed approach to overcome sample size bias. Journal of Microbiological Methods. 75 86-91.

10. ANEXOS

10.1- Programas

```
/*
```

```
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package my.leerlist;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.util.ArrayList;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;

/**
 *
 * @author Luis López
 */
public class LeerListUI extends javax.swing.JFrame {
    private int filaDeLaTabla = 0;
    private String patInicial = "./";

    /**
     * Creates new form LeerListUI
     */
    public LeerListUI() {
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jPanel1 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        abrirArchivo_Button = new javax.swing.JButton();
        nombreArchivo_TextField = new javax.swing.JTextField();
        jLabel2 = new javax.swing.JLabel();
        umbrales_ComboBox = new javax.swing.JComboBox();
        jLabel3 = new javax.swing.JLabel();
        valorDeTDeCorte_TextField = new javax.swing.JTextField();
        jLabel4 = new javax.swing.JLabel();
        cantidadDeIndividuos_TextField = new javax.swing.JTextField();
        corridasAEjecutar_TextField = new javax.swing.JTextField();
        jLabel5 = new javax.swing.JLabel();
        agregar_Button = new javax.swing.JButton();
        jPanel2 = new javax.swing.JPanel();
        jScrollPane1 = new javax.swing.JScrollPane();
        seleccion_Table = new javax.swing.JTable();
        eliminar_Button = new javax.swing.JButton();
        procesar_Button = new javax.swing.JButton();
        jScrollPane2 = new javax.swing.JScrollPane();
        mensajes_TextArea = new javax.swing.JTextArea();
        cantHilos_TextField = new javax.swing.JTextField();
        jLabel6 = new javax.swing.JLabel();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Procesar Archivo(s)  .list");
        setBounds(new java.awt.Rectangle(300, 200, 0, 0));
        setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowOpened(java.awt.event.WindowEvent evt) {
                formWindowOpened(evt);
            }
        });

        jPanel1.setBorder(javax.swing.BorderFactory.createTitledBorder(null, "Selección de
la(s) Muestra(s)", javax.swing.border.TitledBorder.LEFT,
javax.swing.border.TitledBorder.TOP, new java.awt.Font("Arial", 1, 12))); // NOI18N
        jPanel1.setFont(new java.awt.Font("Arial", 1, 12)); // NOI18N

        jLabel1.setText("Archivo: ");
```

```
        abrirArchivo_Button.setText("...");
        abrirArchivo_Button.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                abrirArchivo_ButtonActionPerformed(evt);
            }
        });

        nombreArchivo_TextField.setFont(new java.awt.Font("Arial", 1, 12)); // NOI18N

        jLabel2.setText("Umbral Disimilitudes: ");

        umbrales_ComboBox.setFont(new java.awt.Font("Arial", 1, 12)); // NOI18N

        jLabel3.setText("Valor de T de corte: ");

        valorDeTDeCorte_TextField.setFont(new java.awt.Font("Arial", 1, 12)); // NOI18N
        valorDeTDeCorte_TextField.setHorizontalAlignment(javax.swing.JTextField.CENTER);
        valorDeTDeCorte_TextField.setText("0.0000005");
        valorDeTDeCorte_TextField.addFocusListener(new java.awt.event.FocusAdapter() {
            public void focusLost(java.awt.event.FocusEvent evt) {
                valorDeTDeCorte_TextFieldFocusLost(evt);
            }
        });

        jLabel4.setText("Cantidad de Individuos a simular: ");

        cantidadDeIndividuos_TextField.setFont(new java.awt.Font("Arial", 1, 12)); // NOI18N

cantidadDeIndividuos_TextField.setHorizontalAlignment(javax.swing.JTextField.CENTER);
        cantidadDeIndividuos_TextField.setText("5000000000");
        cantidadDeIndividuos_TextField.addFocusListener(new java.awt.event.FocusAdapter() {
            public void focusLost(java.awt.event.FocusEvent evt) {
                cantidadDeIndividuos_TextFieldFocusLost(evt);
            }
        });

        corridasAEjecutar_TextField.setFont(new java.awt.Font("Arial", 1, 12)); // NOI18N
        corridasAEjecutar_TextField.setHorizontalAlignment(javax.swing.JTextField.CENTER);
        corridasAEjecutar_TextField.setText("10");
        corridasAEjecutar_TextField.addFocusListener(new java.awt.event.FocusAdapter() {
            public void focusLost(java.awt.event.FocusEvent evt) {
                corridasAEjecutar_TextFieldFocusLost(evt);
            }
        });

        jLabel5.setText("Corridas a ejecutar sobre la muestra: ");

        agregar_Button.setText("Agregar");
        agregar_Button.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                agregar_ButtonActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addComponent(jLabel3)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                        .addComponent(valorDeTDeCorte_TextField,
javax.swing.GroupLayout.DEFAULT_SIZE, 93, Short.MAX_VALUE)
                        .addGap(18, 18, 18)
                        .addComponent(jLabel4)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                        .addComponent(cantidadDeIndividuos_TextField,
javax.swing.GroupLayout.DEFAULT_SIZE, 103, Short.MAX_VALUE)
                        .addGap(66, 66, 66))
                    .addGroup(jPanel1Layout.createSequentialGroup()
```

```
.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addGroup(jPanel1Layout.createSequentialGroup()
                                .addComponent(jLabel1)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                                .addComponent(abrirArchivo_Button,
javax.swing.GroupLayout.PREFERRED_SIZE, 25, javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                                .addComponent(nombreArchivo_TextField)
                                .addGap(18, 18, 18)
                                .addComponent(jLabel2)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                                .addComponent(umbrales_ComboBox,
javax.swing.GroupLayout.PREFERRED_SIZE, 64, javax.swing.GroupLayout.PREFERRED_SIZE))
                            .addGroup(jPanel1Layout.createSequentialGroup()
                                .addComponent(jLabel5)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                                .addComponent(corridasAEjecutar_TextField,
javax.swing.GroupLayout.PREFERRED_SIZE, 54, javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                                .addComponent(agregar_Button)))
                        .addGap(10, 10, 10))))
        );
        jPanel1Layout.setVerticalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel1)
                    .addComponent(abrirArchivo_Button)
                    .addComponent(nombreArchivo_TextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel2)
                    .addComponent(umbrales_ComboBox, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel3)
                    .addComponent(valorDeTDeCorte_TextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel4)
                    .addComponent(cantidadDeIndividuos_TextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel5)
                    .addComponent(corridasAEjecutar_TextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(agregar_Button))
                .addGap(19, 19, 19))
        );

        jPanel2.setBorder(javax.swing.BorderFactory.createTitledBorder(null, "Generar
Simulación", javax.swing.border.TitledBorder.LEFT, javax.swing.border.TitledBorder.TOP, new
java.awt.Font("Arial", 1, 12))); // NOI18N

        seleccion_Table.setFont(new java.awt.Font("Tahoma", 1, 12)); // NOI18N
        seleccion_Table.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
```

```
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null},
                {null, null, null, null, null}
            },
            new String [] {
                "Archivo", "Umbral", "T de Corte", "Max Indiv", "Corridas"
            }
        ) {
            Class[] types = new Class [] {
                java.lang.String.class, java.lang.String.class, java.lang.String.class,
java.lang.String.class, java.lang.String.class
            };

            public Class getColumnClass(int columnIndex) {
                return types [columnIndex];
            }
        });
        seleccion_Table.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
        seleccion_Table.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseReleased(java.awt.event.MouseEvent evt) {
                seleccion_TableMouseReleased(evt);
            }
        });
        jScrollPane1.setViewportView(seleccion_Table);

        eliminar_Button.setText("Eliminar");
        eliminar_Button.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                eliminar_ButtonActionPerformed(evt);
            }
        });

        procesar_Button.setText("Procesar");
        procesar_Button.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                procesar_ButtonActionPerformed(evt);
            }
        });

        mensajes_TextArea.setColumns(20);
        mensajes_TextArea.setRows(5);
        jScrollPane2.setViewportView(mensajes_TextArea);

        cantHilos_TextField.setFont(new java.awt.Font("Arial", 1, 12)); // NOI18N
        cantHilos_TextField.setHorizontalAlignment(javax.swing.JTextField.CENTER);
        cantHilos_TextField.setText("02");
        cantHilos_TextField.addFocusListener(new java.awt.event.FocusAdapter() {
            public void focusLost(java.awt.event.FocusEvent evt) {
                cantHilos_TextFieldFocusLost(evt);
            }
        });

        jLabel6.setText("Cantidad de hilos");

        javax.swing.GroupLayout jPanel2Layout = new javax.swing.GroupLayout(jPanel2);
```

```
        jPanel2.setLayout(jPanel2Layout);
        jPanel2Layout.setHorizontalGroup(
            jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 570,
Short.MAX_VALUE)
            .addGroup(jPanel2Layout.createSequentialGroup()
                .addComponent(eliminar_Button)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jScrollPane2)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel2Layout.createSequentialGroup()
                        .addComponent(jLabel6)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                        .addComponent(cantHilos_TextField,
javax.swing.GroupLayout.PREFERRED_SIZE, 38, javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addComponent(procesar_Button,
javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.PREFERRED_SIZE, 125,
javax.swing.GroupLayout.PREFERRED_SIZE)))
        );
        jPanel2Layout.setVerticalGroup(
            jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel2Layout.createSequentialGroup()
                .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 221,
Short.MAX_VALUE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)
                    .addComponent(eliminar_Button, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addGroup(jPanel2Layout.createSequentialGroup()

.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                            .addComponent(cantHilos_TextField,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(jLabel6))
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(procesar_Button, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                    .addComponent(jScrollPane2, javax.swing.GroupLayout.PREFERRED_SIZE, 49,
javax.swing.GroupLayout.PREFERRED_SIZE)))
        );

        jPanel2Layout.linkSize(javax.swing.SwingConstants.VERTICAL, new java.awt.Component[]
{eliminar_Button, procesar_Button});

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(jPanel2, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addGap(5, 5, 5)
                .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE, 119,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jPanel2, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
```

```
        );

        getAccessibleContext().setAccessibleDescription("Permite seleccionar los archivos
.list  a procesar\nademás de otras características.");

        pack();
        setLocationRelativeTo(null);
    }// </editor-fold>

    private void abrirArchivo_ButtonActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:

        JFileChooser fc = new JFileChooser(patInicial);
        fc.setDialogTitle("Elija el archivo a procesar");
        fc.setFileSelectionMode(JFileChooser.FILES_ONLY);
        fc.setApproveButtonText("Seleccionar");
        int res = fc.showOpenDialog(this);
        if(res == JFileChooser.APPROVE_OPTION)
        {
            umbrales_ComboBox.removeAllItems();
            nombreArchivo_TextField.setText("");
            File ar = fc.getSelectedFile();
            String nomArch;
            try {
                nomArch = ar.getCanonicalPath();
                this.nombreArchivo_TextField.setText(nomArch);
                RandomAccessFile archRAF;
                try{
                    archRAF = new RandomAccessFile(nomArch, "r");
                    try {
                        while(archRAF.length() > archRAF.getFilePointer()) {
                            String linea = archRAF.readLine();
                            if(linea.length() > 4)
                                umbrales_ComboBox.addItem(linea.substring(0, 4));
                        }
                        patInicial = ar.getAbsolutePath();
                    } catch (IOException ex) {
                        mensajes_TextArea.append(ex.getMessage() + " - ERROR - lectura del
archivo.\n");
                    }
                    archRAF.close();
                } catch (FileNotFoundException ex) {
                    mensajes_TextArea.append(ex.getMessage() + " - ERROR - archivo no
encontrado.\n");
                }
            } catch (IOException ex) {
                mensajes_TextArea.append(ex.getMessage() + " - ERROR - selecciçon del
archivo.\n");
            }
        }
        if(seleccion_Table.getSelectedRow() != -1) {
            seleccion_Table.removeRowSelectionInterval(seleccion_Table.getSelectedRow(),
seleccion_Table.getSelectedRow());
        }
    }

    private void agregar_ButtonActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        int     col,
                colArch = 0,
                colUmbr = 0,
                fila;
        boolean encontrado = false;

        if( !nombreArchivo_TextField.getText().isEmpty() &&
            !umbrales_ComboBox.getSelectedItem().toString().isEmpty() &&
            filaDeLaTabla < 29) {
            col = 0;
            while(col < 5) {
                switch(seleccion_Table.getColumnName(col)) {
                    case "Archivo"    :
                        colArch = col;
                        break;
                    case "Umbral"     :
                        colUmbr = col;
                }
                col++;
```

```
            }
            fila = 0;

            while(fila < filaDeLaTabla) {
                if(nombreArchivo_TextField.getText().equals(seleccion_Table.getValueAt(fila,
colArch)) &&

umbrales_ComboBox.getSelectedItem().toString().equals(seleccion_Table.getValueAt(fila,
colUmbr))) {
                    encontrado = true;
                }
                fila++;
            }
            if(!encontrado) {
                col = 0;
                while(col < 5) {
                    switch(seleccion_Table.getColumnName(col)) {
                        case "Archivo"    :
                            seleccion_Table.setValueAt(nombreArchivo_TextField.getText(),
filaDeLaTabla, col);
                            break;
                        case "Umbral"     :
                            seleccion_Table.setValueAt(umbrales_ComboBox.getSelectedItem(),
filaDeLaTabla, col);
                            break;
                        case "T de Corte" :
                            seleccion_Table.setValueAt(valorDeTDeCorte_TextField.getText(),
filaDeLaTabla, col);
                            break;
                        case "Max Indiv"  :

seleccion_Table.setValueAt(cantidadDeIndividuos_TextField.getText(), filaDeLaTabla, col);
                            break;
                        case "Corridas"   :

seleccion_Table.setValueAt(corridasAEjecutar_TextField.getText(), filaDeLaTabla, col);
                    }
                    col++;
                }
                filaDeLaTabla++;
            }
        }
        if(filaDeLaTabla > 0)
            procesar_Button.setEnabled(true);
        else
            procesar_Button.setEnabled(false);
        if(seleccion_Table.getSelectedRow() != -1) {
            seleccion_Table.removeRowSelectionInterval(seleccion_Table.getSelectedRow(),
seleccion_Table.getSelectedRow());
        }
    }

    private void eliminar_ButtonActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        int fila = seleccion_Table.getSelectedRow();
        int col;
        while(fila < filaDeLaTabla) {
            col = 0;
            while(col < 5) {
                seleccion_Table.setValueAt(seleccion_Table.getValueAt(fila + 1, col), fila,
col);
                col++;
            }
            fila++;
        }
        col = 0;
        fila = filaDeLaTabla - 1;
        while(col < 5) {
            seleccion_Table.setValueAt("", fila, col);
            col++;
        }
        filaDeLaTabla--;
        if(seleccion_Table.getSelectedRow() != -1) {
            seleccion_Table.removeRowSelectionInterval(seleccion_Table.getSelectedRow(),
seleccion_Table.getSelectedRow());
        }
        corridasAEjecutar_TextField.transferFocus();
```

```
        eliminar_Button.setEnabled(false);
        if(filaDeLaTabla == 0)
            this.procesar_Button.setEnabled(false);
    }

    private void seleccion_TableMouseReleased(java.awt.event.MouseEvent evt) {
        // TODO add your handling code here:
        int fila = seleccion_Table.getSelectedRow();

        if(fila >= filaDeLaTabla ) {
            seleccion_Table.removeRowSelectionInterval(fila, fila);
            corridasAEjecutar_TextField.transferFocus();
            eliminar_Button.setEnabled(false);
        }
        else {
            eliminar_Button.setEnabled(true);
        }
    }

    private void formWindowOpened(java.awt.event.WindowEvent evt) {
        // TODO add your handling code here:
        nombreArchivo_TextField.setEditable(false);
        eliminar_Button.setEnabled(false);
        procesar_Button.setEnabled(false);
        mensajes_TextArea.setEditable(false);
    /// seleccion_Table.setEnabled(false);
    }

    private void procesar_ButtonActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        boolean error = true;
        int col,
            colArch = 0,
            colUmbr = 1,
            colTdeC = 2,
            colMaxI = 3,
            colCorr = 4;
        col = 0;
        String comando = "";
        try {
            comando = "cmd /c del  /f /q .\\tmp\\*.*";
            Runtime.getRuntime().exec(comando).waitFor();
            comando = "cmd /c rm /s /q .\\tmp";
            Runtime.getRuntime().exec(comando).waitFor();
            comando = "cmd /c md  .\\tmp";
            Runtime.getRuntime().exec(comando).waitFor();
        } catch (IOException | InterruptedException  ex) {
            mensajes_TextArea.append(ex.getMessage() + " - ERROR - ejecutando " + comando +
"\n");
        }

        while(col < 5) {
            switch(seleccion_Table.getColumnName(col)) {
                case "Archivo"    :
                    colArch = col;
                    break;
                case "Umbral"     :
                    colUmbr = col;
                    break;
                case "T de Corte" :
                    colTdeC = col;
                    break;
                case "Max Indiv"  :
                    colMaxI = col;
                    break;
                case "Corridas"   :
                    colCorr = col;
            }
            col++;
        }
        String linea = "";
        int fila = 0;
        int ciclo;
        while(fila < filaDeLaTabla) {
            int corridas = Integer.parseInt(seleccion_Table.getValueAt(fila,
colCorr).toString());
            ciclo = 0;
```

```
            while(ciclo < corridas) {
                linea += seleccion_Table.getValueAt(fila, colUmbr).toString() + "\t" +
                         seleccion_Table.getValueAt(fila, colTdeC).toString() + "\t" +
                         seleccion_Table.getValueAt(fila, colMaxI).toString() + "\t" +
                         seleccion_Table.getValueAt(fila, colArch).toString() + "\r\n";
                ciclo++;
            }
            fila++;
            error = false;
        }
        String [] lineasArray = linea.split("\r\n");
        int cantHilos = Integer.parseInt(cantHilos_TextField.getText());
        int numHilo;
        this.setVisible(false);

        Object[] opc1 = {"De Acuerdo."};
        int n = JOptionPane.showOptionDialog(null,
            "El programa comenzará a ejecutar la simulación\n" +
            "Esto puede durar entre varios minutos o unas\n" +
            "   cuantas horas. Una vez que termine le avi-\n" +
            "   sará y terminará de ejecutarse.",
            "Aviso IMPORTANTE.",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.INFORMATION_MESSAGE,
            null,     //do not use a custom Icon
            opc1,  //the titles of buttons
            opc1[0]); //default button title
        this.setEnabled(false);

        RandomAccessFile archRAF;
        ArrayList  procs = new ArrayList(0);
        try{
            for(numHilo = 0; numHilo < cantHilos; numHilo++) {
                String nomArchivo = "./tmp/LeerList" + String.format("%03d", numHilo + 1) +
".txt";
                archRAF = new RandomAccessFile(nomArchivo, "rw");
                archRAF.seek(0);
                archRAF.setLength(0);
                for(ciclo = numHilo; ciclo < lineasArray.length; ciclo = ciclo + cantHilos)
{
                    archRAF.writeBytes(lineasArray[ciclo] + "\r\n");
                }
                archRAF.close();
                comando = "./leerlist.exe  " + nomArchivo;
             ///String comando = String.format("cmd /c dir c:\\*.*  /s > %d.txt", numHilo +
1);
             ///Process q=Runtime.getRuntime().exec (comando);
 ///            procesos.addElement(Runtime.getRuntime().exec(comando));
                procs.add(Runtime.getRuntime().exec(comando));
            }

            for(numHilo = 0; numHilo < cantHilos; numHilo++) {
 ///            ((Process) procesos.get(numHilo)).waitFor();
                ((Process) procs.get(numHilo)).waitFor();
            }
        } catch (Exception  e) {
            mensajes_TextArea.append(e.getMessage() + " - ERROR - ejecutando " + comando +
"\n");
        }
        this.setEnabled(true);
        comando = "./genearinforme.exe  " + String.format("  %03d", cantHilos);
        try {
            Runtime.getRuntime().exec(comando);
        } catch (IOException ex) {
           mensajes_TextArea.append(ex.getMessage() + " - ERROR - ejecutando " + comando +
"\n");
        }
        this.setVisible(true);
        Object []opc2 = {"De Acuerdo."};
        n = JOptionPane.showOptionDialog(null,
                "La simulación ha concluido.\nProceda a ver los resultados",
                "La simulación ha concluído.",
                JOptionPane.YES_NO_OPTION,
                JOptionPane.INFORMATION_MESSAGE,
                null,     //do not use a custom Icon
                opc2,  //the titles of buttons
                opc2[0]); //default button title
```

```
    }

    private void corridasAEjecutar_TextFieldFocusLost(java.awt.event.FocusEvent evt) {
        // TODO add your handling code here:
        try {
            String valor = corridasAEjecutar_TextField.getText();
            int pos = 0;
            while(pos < valor.length()) {
                char c = valor.charAt(pos);
                if(c < '0' || c > '9') {
                    break;
                }
                pos++;
            }
            valor = valor.substring(0, pos);
            int num = Integer.valueOf(valor);
            if(num < 1){
                num = 1;
            }
            if(num > 99){
                num = 99;
            }
            corridasAEjecutar_TextField.setText(String.format("%02d", num));
        } catch (Exception e) {
            corridasAEjecutar_TextField.setText("01");
            mensajes_TextArea.append(e.getMessage() + " - ERROR - cantidad de corridas.\n");
        }
        if(seleccion_Table.getSelectedRow() != -1) {
            seleccion_Table.removeRowSelectionInterval(seleccion_Table.getSelectedRow(),
seleccion_Table.getSelectedRow());
        }
    }

    private void cantidadDeIndividuos_TextFieldFocusLost(java.awt.event.FocusEvent evt) {
        // TODO add your handling code here:  5000000000
        try {
            String valor = this.cantidadDeIndividuos_TextField.getText();
            int pos = 0;
            while(pos < valor.length()) {
                char c = valor.charAt(pos);
                if(c < '0' || c > '9') {
                    break;
                }
                pos++;
            }
            valor = valor.substring(0, pos);
            Long num = Long.valueOf(valor);
            if(num < 1000L) { //000L || num > 5000000000L)
                num = 1000L;
            }
            if(num > 5000000000L) { //000L || num > 5000000000L)
                num = 5000000000L;
            }
            cantidadDeIndividuos_TextField.setText(String.format("%010d", num));
        } catch (Exception e) {
            cantidadDeIndividuos_TextField.setText("5000000000");
            mensajes_TextArea.append(e.getMessage() + " - ERROR - cantidad de
individuos.\n");
        }
        if(seleccion_Table.getSelectedRow() != -1) {
            seleccion_Table.removeRowSelectionInterval(seleccion_Table.getSelectedRow(),
seleccion_Table.getSelectedRow());
        }
    }

    private void valorDeTDeCorte_TextFieldFocusLost(java.awt.event.FocusEvent evt) {
        // TODO add your handling code here:  0.0000005
        try {
            String valor = this.valorDeTDeCorte_TextField.getText();
            int pos = 0;
            while(pos < valor.length()) {
                char c = valor.charAt(pos);
                if(pos != 1) {
                    if(c < '0' || c > '9') {
                        break;
                    }
```

```
                }
                if(pos == 1) {
                    if(c != '.') {
                        break;
                    }
                }
                pos++;
            }
            valor = valor.substring(0, pos);
            double num = Double.valueOf(valor);
            if(num >= 0.005) {
                num = 0.005;
            }
            if(num < 0.0000005) {
                num = 0.0000005;
            }
            valorDeTDeCorte_TextField.setText(String.format("%09.7f", num).replace(',',
'.'));
        } catch (Exception e) {
            valorDeTDeCorte_TextField.setText("0.0000500");
            mensajes_TextArea.append(e.getMessage() + " - ERROR - T de Corte.\n");
        }
        if(seleccion_Table.getSelectedRow() != -1) {
            seleccion_Table.removeRowSelectionInterval(seleccion_Table.getSelectedRow(),
seleccion_Table.getSelectedRow());
        }
    }

    private void cantHilos_TextFieldFocusLost(java.awt.event.FocusEvent evt) {
        // TODO add your handling code here:
        try {
            String valor = cantHilos_TextField.getText();
            int pos = 0;
            while(pos < valor.length()) {
                char c = valor.charAt(pos);
                if(c < '0' || c > '9') {
                    break;
                }
                pos++;
            }
            valor = valor.substring(0, pos);
            int num = Integer.valueOf(valor);
            if(num < 1){
                num = 1;
            }
            if(num > 99){
                num = 99;
            }
            cantHilos_TextField.setText(String.format("%02d", num));
        } catch (Exception e) {
            cantHilos_TextField.setText("01");
            mensajes_TextArea.append(e.getMessage() + " - ERROR - cantidad de hilos.\n");
        }
        if(seleccion_Table.getSelectedRow() != -1) {
            seleccion_Table.removeRowSelectionInterval(seleccion_Table.getSelectedRow(),
seleccion_Table.getSelectedRow());
        }
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional)
">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look
and feel.
         * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
```

```
                }
            }
        } catch (ClassNotFoundException | InstantiationException | IllegalAccessException |
                 javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(LeerListUI.class.getName()).log(java.util.logging.Level.S
EVERE, null, ex);
        }
        //</editor-fold>

        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(() -> {
            new LeerListUI().setVisible(true);
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton abrirArchivo_Button;
    private javax.swing.JButton agregar_Button;
    private javax.swing.JTextField cantHilos_TextField;
    private javax.swing.JTextField cantidadDeIndividuos_TextField;
    private javax.swing.JTextField corridasAEjecutar_TextField;
    private javax.swing.JButton eliminar_Button;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JTextArea mensajes_TextArea;
    private javax.swing.JTextField nombreArchivo_TextField;
    private javax.swing.JButton procesar_Button;
    private javax.swing.JTable seleccion_Table;
    private javax.swing.JComboBox umbrales_ComboBox;
    private javax.swing.JTextField valorDeTDeCorte_TextField;
    // End of variables declaration
}
```

Código C de la aplicación leerList.exe: (main.c) - Volver

```
#include "main.h"

int main(int cArg, char **vArg)
{
    int          nroArchLeerList = -1,  /// da el número de hilo
                 retVal;
    char         leerList[9],
                 dotTxt[5];
    FILE        *fpLeerList,
                *fpTemp,
                *fpSimulacion;
    char         linea[500],
                *aux,
                *diversidad,  /// umbral de disimilitudes
                *tTDeCorte,
                *nroMaxInd,
                *nomArchList,
                 nomArchTmp[500];
    t_lista      muestra;


    if(cArg != 2)
        return ERR_CANTARGMAIN;
    sscanf(vArg[1] + 6, "%8s%3d%4s", leerList, &nroArchLeerList, dotTxt);
    if(nroArchLeerList < 1 ||
       strcmpi(leerList, "leerlist") != 0 ||
```

Código C de la aplicación leerList.exe: (main.c) - Volver

```
       strcmpi(dotTxt, ".txt") != 0 ||
       strlen(vArg[1]) != 21)
        return ERR_PARAMMAIN;
    sprintf(nomArchTmp, "./tmp/Simulacion%03d.csv", nroArchLeerList);
    if(!abrirArchivo(&fpSimulacion, nomArchTmp, "wt", CON_MSJ))
    {
        return ERR_ABRIENDOARCH;
    }
    fprintf(fpSimulacion, "\tCant.Clus.:\tSingletones:\tTot. Indiv.\tT.\tT Transc.\n");

    sprintf(nomArchTmp, "./tmp/Temp%03d.txt", nroArchLeerList);
    if(!abrirArchivo(&fpLeerList, vArg[1], "rt", CON_MSJ))
        return ERR_ABRIENDOARCH;
    while(fgets(linea, sizeof(linea), fpLeerList))
    {
        if((aux = strchr(linea, '\n')) == NULL)
            return ERR_LEYENDOLIST;
        *aux = '\0';
        diversidad = linea;
        if((tTDeCorte = strchr(diversidad + 1, '\t')) == NULL)
            return ERR_LEYENDOLIST;
        *tTDeCorte = '\0';
        tTDeCorte++;
        if((nroMaxInd = strchr(tTDeCorte + 1, '\t')) == NULL)
            return ERR_LEYENDOLIST;
        *nroMaxInd = '\0';
        nroMaxInd++;
        if((nomArchList = strchr(nroMaxInd + 1, '\t')) == NULL)
            return ERR_LEYENDOLIST;
        *nomArchList = '\0';
        nomArchList++;
        if((aux = strchr(nomArchList, '\t')) != NULL)
            return ERR_LEYENDOLIST;
        if(!abrirArchivo(&fpTemp, nomArchTmp, "wb", CON_MSJ))
        {
            fclose(fpLeerList);
            return ERR_ABRIENDOARCHTMP;
        }
        if((retVal = abrirYParsear(nomArchList, diversidad,
                                   tTDeCorte, nroMaxInd,
                                   fpTemp)) != 0)
            return retVal;
        fprintf(fpSimulacion,
                "Archivo:        %s\n"
                "Umb Disimilit.: %s\n"
                "T. de Corte:    %s\n"
                "Max Individuos: %s\n",
                nomArchList, diversidad, tTDeCorte, nroMaxInd);
        fclose(fpTemp);
        if(!abrirArchivo(&fpTemp, nomArchTmp, "rt", CON_MSJ))
        {
            fclose(fpLeerList);
            return ERR_ABRIENDOARCHTMP;
        }
        crearLista(&muestra);
        if(!leerTemporario(fpTemp, &muestra))
        {
///         fprintf(stderr, "ERROR - Contando individuos\n");
            return ERR_CONTANDOIND;
        }
        fclose(fpTemp);
        if(!calcularFrecuencias(&muestra, fpSimulacion))
        {
           fprintf(stderr, "Lista Vacia\n");
           return 4;
        }

        if(!agregarMuestras(&muestra, fpSimulacion, tTDeCorte,
                            nroMaxInd, nroArchLeerList))
        {
           fprintf(stderr, "ERROR - en agregar muestras\n");
           return 8;
        }
        fflush(fpSimulacion);
        vaciarLista(&muestra);
```

Código C de la aplicación leerList.exe: (main.c) - Volver

```
    }
    fclose(fpLeerList);
    fclose(fpSimulacion);

    return 0;
}
```

Código C de la aplicación leerList.exe: (main.h) - Volver

```
#ifndef MAIN_H_
#define MAIN_H_

#include <stdio.h>
#include <stdlib.h>

#include "funciones.h"
#include "rand.h"
#include "lista.h"

#define ERR_CANTARGMAIN             1
#define ERR_PARAMMAIN               2
#define ERR_ABRIENDOARCH            4
#define ERR_ABRIENDOARCHTMP         8
#define ERR_LEYENDOLIST             16
#define ERR_PROCSIMULA              32
#define ERR_CONTANDOIND             64


//#define ARCH_ENT        ".\\S85.phy.cgi.fn.list"
//#define ARCH_ENT        ".\\datosc\\S86.phy.cgi.fn.list"

#define ARCH_TEMP       ".\\tmp\\tmp%03d.csv"
/* comenzando con el archivo tmp00.csv */

//#define INFO_1          ".\\1merInfo.csv"
//#define INFO_2          ".\\Info.csv"

#endif
```

Código C de la aplicación leerList.exe: (funciones.c) - Volver

```
#include "main.h"
#include "funciones.h"
#include "rand.h"

/**
 * función:    crearTemp
 * recibe:     fpEnt (E) - Archivo de entrada proveniente del secuenciador
 * devuelve:   0 (cero) - en caso de fracaso en su cometido
 *             numArch (distinto de cero) - cantidad de archivos creados, uno
 *                por cada "fila"
 * precond.:   el archivo de entrada (fpEnt) debe estar correctamente abierto
 * observ.:    utiliza ARCH_TEMP para directorio/nombre-archivo
 */
int crearTemp(FILE *fpEnt)
{
   FILE       *fpTemp = NULL;
   char        linea[4096],
               nomArch[MAX_PATH];
   int         cambioDeArch = 1,
               numArch = 0;

   while(fgets(linea, sizeof(linea), fpEnt))
   {
      if(cambioDeArch)
      {
         if(fpTemp)
            fclose(fpTemp);
         sprintf(nomArch, ARCH_TEMP, numArch);
         if(!abrirArchivo(&fpTemp, nomArch, "wt", CON_MSJ))
            return 0;
```

Código C de la aplicación leerList.exe: (funciones.c) - Volver

```
         numArch++;
         cambioDeArch = 0;
      }

      cambioDeArch = reemplazarNueLinXNull(linea);
      reemplazarTabXNueLinYComaXTab(linea);
      fprintf(fpTemp, linea);
   }
   if(fpTemp)
      fclose(fpTemp);

   return numArch;
}


/**
 * función:    reemplazarTabXNueLinYComaXTab
 * recibe:     linea (E/S) - Cadena a modificar
 * devuelve:   n/a
 * precond.:   debe recibir una cadena terminada en '\0'
 * observ.:
 */
void reemplazarTabXNueLinYComaXTab(char *linea)
{
   char *aux = linea;

   while((aux = strchr(aux, '\t')) != NULL)
   {
      *aux = '\n';
      aux++;
   }
   aux = linea;
   while((aux = strchr(aux, ',')) != NULL)
   {
      *aux = '\t';
      aux++;
   }
}


/**
 * función:    reemplazarNueLinXNull
 * recibe:     linea (E/S) - Cadena a modificar
 * devuelve:   indicador de que encontró y reemplazó o cero
 * precond.:   debe recibir una cadena terminada en '\0'
 * observ.:
 */
int reemplazarNueLinXNull(char *linea)
{
   if((linea = strchr(linea, '\n')) != NULL)
   {
      *linea = '\0';
      return 1;
   }
   return 0;
}


/**
 * funcion:    leerTemporario
 * recibe:     queTemp (E)  - -1 = unique / 0 = 0.00 / 1 = 0.01 / etc.
 *             p      (E/S) - lista en la que almacena la información
 * devuelve:   0 (cero) en caso de fracaso ó 1 (uno)
 * precond.:   que la lista esté creada como lista vacía
 * obser.:
 *//**
int leerTemporario(int queTemp, t_lista *p)*/
int leerTemporario(FILE *fp, t_lista *p)
{
   char     linea[500000],
           *aux;
   int      cantIndividuos,
            retVal = 0;
   t_info   info;
```

Código C de la aplicación leerList.exe: (funciones.c) - Volver

```
   /* primera línea */
   if(!fgets(linea, sizeof(linea), fp) ||
      (aux = strchr(linea, '\n')) == NULL)
      goto Salida;
   *aux = '\0';
   strcpy(p->corte, linea);
   /* segunda línea */
   if(!fgets(linea, sizeof(linea), fp) ||
      (aux = strchr(linea, '\n')) == NULL)
      goto Salida;
   *aux = '\0';
   sscanf(linea, "%ld", &p->clusters);
   /* procesa los cluster determinando cantidad de individuos en el cluster */
   while(fgets(linea, sizeof(linea), fp))
   {
      if((aux = strchr(linea, '\n')) == NULL)
         goto Salida;
      *aux = '\0';
      /* contar individuos del cluster */
      aux = linea;
      cantIndividuos = 0;
      while((aux = strchr(aux, '\t')) != NULL)
      {
         cantIndividuos++;
         aux++;
      }
      info.cantIndividuos = ++cantIndividuos;
      info.cantClusters = 1;
      if(!insertarEnOrden(p, &info))
         goto Salida;
   }
   retVal = 1;
Salida:

   return retVal;
}


/**
 * funcion:    leerMuestra
 * recibe:     fpEnt (E) - archivo temporario a procesar
 *             p (E/S) - lista en la que almacena la información
 * devuelve:   0 (cero) en caso de fracaso ó 1 (uno)
 * precond.:   que el archivo esté abierto y
 *                que la lista esté creada como lista vacía
 * obser.:
 */
int leerMuestra(FILE *fp, t_lista *p)
{
   return 1;
}


/**
 * funcion:    agregarMuestras
 * recibe:     la lista
 * devuelve:   1 (n/a)
 * precond.:   que la lista esté cargada con la info del secuenciador
 * obser.:
 */
int agregarMuestras(t_lista *p, FILE *fpSimul,
                    const char *tTDeCorte, const char *nroMaxInd,
                    int numHilo)
{
    long         idNum = 123456 + (numHilo * 1331) + time(NULL);
    t_rand3      rand3;
    long         idNumBis = 154326 + (numHilo * 1331) + time(NULL);
    t_rand3      rand3Bis;
    double       maxTt;
    long long    maxIn;
    int          marca1 = 1,
                 marca2 = 1,
                 marca3 = 1,
                 marca4 = 1,
```

Código C de la aplicación leerList.exe: (funciones.c) - Volver

```
                  marca5 = 1,
                  marca6 = 1,
                  marca7 = 1,
                  marca8 = 1,
                  marca  = 0;
    maxTt = atof(tTDeCorte);
    sscanf(nroMaxInd, "%I64d", &maxIn);

   inicializarRand3(idNum, &rand3);
   inicializarRand3(idNumBis, &rand3Bis);
///while(p->singletones && p->tT)
   while(marca8)
   {
/**(1)cicloDos = 100;
      while(cicloDos-- && p->singletones && p->tT)
      {
/ **      if(generarRan3(&rand3) < obtenerT(p))  */
         if(generarRan3(&rand3) <= obtenerT(p))
            agregarSingleton(p);
         else
            agregarIndivQueEstaba(p, generarRan3(&rand3Bis));
///      calcularFrecuencias(p, NULL);
///(1)}
         if(marca1 && p->tT < 0.0300005)
         {
            marca1 = 0;
            marca = 1;
         }
         if(marca2 && p->tT < 0.0100005)
         {
            marca2 = 0;
            marca = 1;
         }
         if(marca3 && p->tT < 0.0010005)
         {
            marca3 = 0;
            marca = 1;
         }
         if(marca4 && p->tT < 0.0001005)
         {
            marca4 = 0;
            marca = 1;
         }
         if(marca5 && p->tT < 0.0000105)
         {
            marca5 = 0;
            marca = 1;
         }
         if(marca6 && p->tT < 0.0000015)
         {
            marca6 = 0;
            marca = 1;
         }
         if(marca7 && p->tT < 0.0000005)
         {
            marca7 = 0;
            marca8 = 0;
            marca = 1;
         }
         if(marca8 && (p->tT < maxTt  ||  p->totIndiv > maxIn))
         {
            marca8 = 0;
            marca = 1;
         }

         if(marca)
         {
            marca = 0;
 /**/       calcularFrecuencias(p, fpSimul);  /**/
         }
         else
            calcularFrecuencias(p, NULL);
   }
/**calcularFrecuencias(p, fpSimul);  **/
   return 1;
```

Código C de la aplicación leerList.exe: (funciones.c) - Volver

```
}

/**
 *
 */
int abrirYParsear(const char *nomArchList, const char *diversidad,
                  const char *tTDeCorte,   const char *nroMaxInd,
                  FILE       *fpTemp)
{
    FILE       *fpList;
    char        buffer[20000], //[65537], ///
               *aux;
    long long   leidos = -1,
                totalLeidos = 0;
    int         encontrado = 0,
                hayFinLinea = 0;

    if(!abrirArchivo(&fpList, nomArchList, "rb", CON_MSJ))
    {
        return ERR_PROCSIMULA;
    }
    while((leidos = fread(buffer, 1, sizeof(buffer) - 1, fpList)) != 0)
    {
        hayFinLinea = 0;
        *(buffer + leidos) = '\0';
        if((aux = strchr(buffer, '\r')) != NULL)
        {
            *aux = '\0';
            leidos = (int)(aux - buffer) + 2;
            hayFinLinea = 1;
        }
        totalLeidos += leidos;
        fseek(fpList, totalLeidos, SEEK_SET);
        if(!strnicmp(buffer, diversidad, strlen(diversidad)) || encontrado)
        {

            encontrado++;
            buscarYReemplazar(buffer, '\t', '\n');
            buscarYReemplazar(buffer, ',', '\t');
            fwrite(buffer, 1, leidos - (hayFinLinea ? 2 : 0), fpTemp);
            fflush(fpTemp);
            if(hayFinLinea)
                break;
        }
    }
    return 0;
}


/**
 *
 */
void buscarYReemplazar(char *buffer, char que, char por)
{
    char *aux = buffer;
    while((aux = strchr(aux, que)) != NULL)
    {
        *aux = por;
        aux++;
    }
}
```

Código C de la aplicación leerList.exe: (funciones.h) - Volver

```
#ifndef FUNCIONES_H_
#define FUNCIONES_H_

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
```

Código C de la aplicación leerList.exe: (funciones.h) - Volver

```
#include "lista.h"



#define CON_MSJ         1

#define abrirArchivo( X, Y, Z, Q )                                          \
            ( (*(X) = fopen(Y, Z)) == NULL  ?                               \
                  ( Q == CON_MSJ ?                                          \
                     !fprintf(stderr,                                       \
                             "ERROR - abriendo \"%s\" en modo \"%s\"\n",    \
                             Y, Z) | 0  :                                   \
                     0 ) :                                                  \
                  1 )


int crearTemp(FILE *fpEnt);

void reemplazarTabXNueLinYComaXTab(char *linea);

int reemplazarNueLinXNull(char *linea);

int contarIndividuos(int cantTemp);

int leerMuestra(FILE *fp, t_lista *p);

///int leerTemporario(int queTemp, t_lista *p);
int leerTemporario(FILE *fp, t_lista *p);

int agregarMuestras(t_lista *p, FILE *fpSimul,
                    const char *tTDeCorte, const char *nroMaxInd,
                    int numHilo);

int abrirYParsear(const char *nomArchList, const char *diversidad,
                  const char *tTDeCorte,   const char *nroMaxInd,
                  FILE       *ftTemp);

void buscarYReemplazar(char *buffer, char que, char por);

#endif
```

Código C de la aplicación leerList.exe: (lista.c) - Volver

```
#include "lista.h"

/**
 * funcion:
 * recibe:
 * devuelve:
 * precond.:
 * obser.:
 */
/**
 * funcion:    crearLista
 * recibe:     la lista
 * devuelve:   (n/a)
 * precond.:   (n/a)
 * obser.:     crea la lista como lista vacía
 */
void crearLista(t_lista *p)
{
   p->pri         = NULL;
   *p->corte      = '\0';
   p->cantNodos   = 0;
   p->clusters    = 0;
   p->singletones = 0;
   p->totIndiv    = 0;
   p->tT          = 0.0;
   p->tiempoIni   = time(NULL);
}

void vaciarLista(t_lista *p)
```

Código C de la aplicación leerList.exe: (lista.c) - Volver

```
{
    t_nodo *aux;
    while(p->pri)
    {
        aux = p->pri;
        p->pri = aux->sig;
        free(aux);
    }
}

/**
 * funcion:    insertarEnOrden
 * recibe:     la lista
 *             la info del secuenciador
 * devuelve:   CLA_DUP y TODO_BIEN si es exitosa
 *             SIN_MEM en caso de fracaso
 * precond.:   que la lista esté inicializada
 * obser.:
 */
int  insertarEnOrden(t_lista *p, const t_info *d/*,
                     int (*comp)(const t_info *, const t_info *),
                     void (*acum)(t_info *, const t_info *)*/)
{
   t_nodo **act = &p->pri,
          *nue;

   if(d->cantIndividuos == 1)
      p->singletones++;
   p->totIndiv += d->cantIndividuos;
/* p->clusters += d->cantClusters;  */
   while(*act && (*act)->info.cantIndividuos < d->cantIndividuos)
      act = &(*act)->sig;
   if(*act && (*act)->info.cantIndividuos == d->cantIndividuos)
   {
      (*act)->info.cantClusters++;
      return CLA_DUP;
   }
   nue = (t_nodo *)malloc(sizeof(t_nodo));
   if(nue == NULL)
      return SIN_MEM;
   nue->info = *d;
   nue->sig = *act;
   *act = nue;
   p->cantNodos++;
   return TODO_BIEN;
}


/**
 * funcion:    calcularFrecuencias
 * recibe:     la lista
 *             stream (de texto) de salida de la información de las poblaciones
 * devuelve:   1 (n/a)
 * precond.:   que la lista no esté vacía
 * obser.:     calcula la frecuencia de cada nodo y la sumatoria de frecuencias
 * NOTA:       hace una salida impresa por la stream que recibe por argumento
 */
int calcularFrecuencias(t_lista *p, FILE *fpStream)
{
   t_nodo *act = p->pri;
   double sumFrecuen = 0.0;

/**if(fpStream)
      fprintf(fpStream,
///            "%s\t%ld clusters\n",
///            "\t%ld\t",
               "\t%ld",
               p->clusters); / **/
   if(act)
   {
 /**  if(fpStream)
         fprintf(fpStream, "Indiv.\tClust.\tFrecuen.\tSum.Frec.\n");  / **/
      while(act->sig)
      {
         sumFrecuen += act->info.frecuencia =
```

Código C de la aplicación leerList.exe: (lista.c) - Volver

```
   /**         act->info.cantClusters / (double)p->clusters; */
               act->info.cantClusters / (double)p->totIndiv;
               /** debo dividir por cantidad de individuos */
         act->info.sumFrecuen = sumFrecuen;
   /**   if(fpStream)
            fprintf(fpStream,
                    "%ld\t%ld\t%lf\t%lf\n",
                    act->info.cantIndividuos,
                    act->info.cantClusters,
                    act->info.frecuencia,
                    act->info.sumFrecuen);   / **/
         act = act->sig;
      }
      act->info.frecuencia =
            act->info.cantClusters / (double)p->clusters;
      act->info.sumFrecuen = 1.0;
 /**  if(fpStream)
         fprintf(fpStream,
              "%ld\t%ld\t%lf\t%lf\n",
              act->info.cantIndividuos,
              act->info.cantClusters,
              act->info.frecuencia,
              act->info.sumFrecuen);   / **/
   }

   p->tT = p->singletones / (double)p->totIndiv;
   if(fpStream)
      fprintf(fpStream,
              "\t%ld\t%ld\t%I64d\t%lf\t%ld\n",
              p->clusters, p->singletones,
              p->totIndiv, p->tT,
              time(NULL) - p->tiempoIni);

   return 1;
}



/**
 * funcion:    agregarSingleton
 * recibe:     la lista
 * devuelve:   TODO_BIEN en caso de éxito
 *             LISTA_VACIA / SIN_MEM en caso de fracaso
 * precond.:   que la lista ya tenga información cargada
 * obser.:
 */
int agregarSingleton(t_lista *p)
{
   t_nodo **aux = &p->pri;
   if(*aux == NULL)
      return LISTA_VACIA;
   if((*aux)->info.cantIndividuos == 1)
   {
      (*aux)->info.cantClusters++;
   }
   else
   {
      t_nodo *nue = (t_nodo *)malloc(sizeof(t_nodo));
      if(nue == NULL)
         return SIN_MEM;
      nue->info.cantClusters = 1;
      nue->info.cantIndividuos = 1;
      nue->sig = *aux;
      *aux = nue;
      p->cantNodos++;
   }
   p->clusters++;
   p->singletones++;
   p->totIndiv++;
   return TODO_BIEN;
}


/**
```

Código C de la aplicación leerList.exe: (lista.c) - Volver

```
 * funcion:     agregarIndivQueEstaba
 * recibe:      t_lista *p - la lista
 *              double frec -
 * devuelve:    TODO_BIEN, de lo contrario
 *              LISTA_VACIA o SIN_MEM
 * precond.:    que la lista esté cargada
 * obser.:
 */
int agregarIndivQueEstaba(t_lista *p, double frec)
{
   t_nodo **act = &p->pri;
   int      cantIndividuos;
   if(*act == NULL)
      return LISTA_VACIA;
   while((*act)->sig && (*act)->info.sumFrecuen < frec)
      act = &(*act)->sig;
   if((*act)->info.cantIndividuos == 1)
      p->singletones--;
   (*act)->info.cantClusters--;
   cantIndividuos = (*act)->info.cantIndividuos + 1;
   if((*act)->info.cantClusters == 0)
   {
      /** hay que eliminar el nodo */
      t_nodo *aux = *act;
      *act = (*act)->sig;
      free(aux);
      p->cantNodos--;
   }
   while(*act && (*act)->info.cantIndividuos < cantIndividuos)
      act = &(*act)->sig;
   if(*act && (*act)->info.cantIndividuos == cantIndividuos)
   {
      (*act)->info.cantClusters++;
   }
   else
   {
      t_nodo *nue = (t_nodo *)malloc(sizeof(t_nodo));
      if(nue == NULL)
         return SIN_MEM;
      nue->info.cantIndividuos = cantIndividuos;
      nue->info.cantClusters = 1;
      nue->sig = *act;
      *act = nue;
      p->cantNodos++;
   }
   p->totIndiv++;
   return TODO_BIEN;
}
```

Código C de la aplicación leerList.exe: (lista.h) - Volver

```
#ifndef LISTA_H_
#define LISTA_H_

#include <stdlib.h>
#include <stdio.h>
#include <time.h>

#define CLA_DUP         2
#define TODO_BIEN       1
#define SIN_MEM         0

#define LISTA_VACIA     -1


typedef struct
{
   long           cantIndividuos,   /**  con esta cantIndividuos,        ... */
                  cantClusters;     /**  hay esta cantCluster,           ... */
   double         frecuencia,       /**  con frecuencia de aparición     ... */
                  sumFrecuen;       /**  y sumFrecuencia                    */
} t_info;
```

Código C de la aplicación leerList.exe: (lista.h) - Volver

```
typedef struct s_nodo
{
   t_info         info;
   struct s_nodo *sig;
} t_nodo;


typedef struct s_lista
{
   t_nodo        *pri;
   char           corte[10];
   long           cantNodos,
                  clusters,
                  singletones;
   long long      totIndiv;
   double         tT;
   long           tiempoIni;
} t_lista;


void crearLista(t_lista *p);


void vaciarLista(t_lista *p);


int  insertarEnOrden(t_lista *p, const t_info *d);


int calcularFrecuencias(t_lista *p, FILE *fpStream);


/** reemplazada por la anterior
int recalcularFrecuencia(t_lista *p);
 */


#define obtenerT(X)        ( ( X )->tT )


int agregarSingleton(t_lista *p);


int agregarIndivQueEstaba(t_lista *p, double frec);


#endif
```

Código C de la aplicación leerList.exe: (rand.c) - Volver

```
#include "rand.h"


/**
 * funcion:    ran3
 * recibe:     long *idum - una semilla que permite reiniciar el generador
 * devuelve:   double un número al azar
 * precond.:   (n/a)
 * obser.:     algoritmo de Knuth tomado de Numerical Recipes in C
 */
double ran3(long *idum)
{
   static int inext,
              inextp;
   static long ma[56]; /* The value 56 (range ma[1..55]) is special and
                        * should not be modified; see Knuth.
                        */
   static int iff = 0;
   long mj,
        mk;
```

Código C de la aplicación leerList.exe: (rand.c) - Volver

```
   int i,
       ii,
       k;
   if (*idum < 0 || iff == 0)
   {
                        /* Initialization. */
      iff = 1;
      mj = labs(MSEED - labs(*idum)); /* Initialize ma[55] using the seed  */
      mj %= MBIG;                     /* idum and the large number MSEED. */
      ma[55] = mj;
      mk = 1;
      for(i = 1; i <= 54; i++)
      {
                                      /* Now initialize the rest of the table, */
         ii = (21 * i) % 55;          /* in a slightly random order, */
         ma[ii] = mk;                 /* with numbers that are not especially random. */
         mk = mj - mk;
         if (mk < MZ)
            mk += MBIG;
         mj=ma[ii];
      }
      for(k = 1; k <= 4; k++)       /* We randomize them by "warming up the generator." */
         for(i = 1; i <= 55; i++)
         {
            ma[i] -= ma[1 + (i + 30) % 55];
            if(ma[i] < MZ)
               ma[i] += MBIG;
         }
      inext = 0;            /* Prepare indices for our first generated number. */
      inextp = 31;          /* The constant 31 is special; see Knuth. */
      *idum = 1;
   }
   /* Here is where we start, except on initialization. */
   if(++inext == 56)
      inext = 1;                    /* Increment inext and inextp, wrapping around */
   if(++inextp == 56)
      inextp = 1;                   /* 56 to 1. */
   mj = ma[inext] - ma[inextp];     /* Generate a new random number subtractively.  */
   if(mj < MZ)
      mj += MBIG;                   /* Be sure that it is in range. */
   ma[inext] = mj;                  /* Store it, */
   return mj * FAC;                 /* and output the derived uniform deviate. */
}




void inicializarRand3(long idum, t_rand3 *p)
{
   long  mk,
         mj;
   int   i,
         ii,
         k;
                        /* Initialization. */
   mj = labs(MSEED - labs(idum)); /* Initialize ma[55] using the seed  */
   mj %= MBIG;                    /* idum and the large number MSEED. */
   p->ma[55] = mj;
   mk = 1;
   for(i = 1; i <= 54; i++)
   {
                                /* Now initialize the rest of the table, */
      ii = (21 * i) % 55;       /* in a slightly random order, */
      p->ma[ii] = mk;           /* with numbers that are not especially random. */
      mk = mj - mk;
      if (mk < MZ)
         mk += MBIG;
      mj = p->ma[ii];
   }
   for(k = 1; k <= 4; k++)      /* We randomize them by "warming up */
      for(i = 1; i <= 55; i++) /* the generator." */
      {
         p->ma[i] -= p->ma[1 + (i + 30) % 55];
         if(p->ma[i] < MZ)
```

Código C de la aplicación leerList.exe: (rand.c) - Volver

```
            p->ma[i] += MBIG;
      }
   p->inext = 0;           /* Prepare indices for our first generated number. */
   p->inextp = 31;         /* The constant 31 is special; see Knuth. */
}

double generarRan3(t_rand3 *p)
{
   long  mj;

   /* Here is where we start, except on initialization. */
   if(++p->inext == 56)
      p->inext = 1;              /* Increment inext and inextp, */
   if(++p->inextp == 56)         /* wrapping around 56 to 1.    */
      p->inextp = 1;
   mj = p->ma[p->inext] - p->ma[p->inextp]; /* Generate a new random number */
                                           /* subtractively.  */
   if(mj < MZ)
      mj += MBIG;                /* Be sure that it is in range. */
   p->ma[p->inext] = mj;         /* Store it, */
   return mj * FAC;              /* and output the derived uniform deviate. */
}
```

Código C de la aplicación leerList.exe: (rand.h) - Volver

```
#ifndef RAND_H_
#define RAND_H_

#include <stdlib.h>       /* Change to math.h in K&R C. */
#define MBIG                  1000000000
#define MSEED                 161803398
#define MZ                    0
#define FAC                   ( 1.0 / (double)MBIG )

/* According to Knuth, any large MBIG, and any smaller (but still large)
 * MSEED can be substituted for the above values.
 */


double ran3(long *idum);
double ran3Bis(long *idum);
/* Returns a uniform random deviate between 0.0 and 1.0.
 * Set idum to any negative value to
 * initialize or reinitialize the sequence.
 */

typedef struct
{
   int   inext,
         inextp;
   long  ma[56];  /** The value 56 (range ma[1..55]) is special and
                   * should not be modified; see Knuth.                  */
} t_rand3;


void inicializarRand3(long idnum, t_rand3 *p);


double generarRan3(t_rand3 *p);


#endif
```

Volver

Código C de la aplicación GenerarInforme.exe: (main.c) - Volver

```
#include "main.h"

int main(int cArg, char **vArg)
{
    int         nroHilos = -1,
                nroHiloActual = 1,
                ciclo;
    FILE       *fpSimulacion,
               *fpSimulacionR;
    char        linea[500],
               *aux,
                nomArchSim[500],
                primeraLinea[500];
/**             prefijos2a4[] = { "Diversida\vT de Cort\vMax Indiv" },
 *              prefijoabuscar[10];
 *              primeraLinea[] =
 *                      { "\tCant.Clus.:\tSingletones:\tTot. Indiv.\tT\n" };
 */
    t_lista     lista;
    t_info      info,
                infoAnt,
                infoVacia = { };
    t_reg       reg;

    crearLista(&lista);

    if(cArg != 2)
        return ERR_CANTARGMAIN;

    sscanf(vArg[1], "%3d", &nroHilos);

    while(nroHiloActual <= nroHilos)
    {
        info = infoVacia;
        sprintf(nomArchSim, "./tmp/Simulacion%03d.csv", nroHiloActual);
        if(!abrirArchivo(&fpSimulacion, nomArchSim, "rt", CON_MSJ))
        {
            break;
        }
        fgets(primeraLinea, sizeof(primeraLinea), fpSimulacion);
        fgets(linea, sizeof(linea), fpSimulacion);
        ciclo = -1;
        while(!feof(fpSimulacion))
        {
            if((aux = strchr(linea, '\n')) != NULL)
                *aux = '\0';
            if(ciclo == -1)
                strncpy(info.nomArch, linea, sizeof(info.nomArch));
            else
            {
                if(ciclo > 2 && *linea != '\t')
                {
                    insertarEnOrden(&lista, &info);
                    info = infoVacia;
                    ciclo = -1;
                    continue;
                }
                strncpy(info.info[ciclo], linea, sizeof(info.info[0]));
            }
            ciclo++;
            fgets(linea, sizeof(linea), fpSimulacion);
        }
        insertarEnOrden(&lista, &info);
        fclose(fpSimulacion);
        nroHiloActual++;
    }


    sprintf(nomArchSim, "Simulacion-.csv");
    if(!abrirArchivo(&fpSimulacion, nomArchSim, "wt", CON_MSJ))
    {
        return ERR_ABRIENDOARCH;
    }
```

Código C de la aplicación GenerarInforme.exe: (main.c) - Volver

```
    sprintf(nomArchSim, "SimulacionR.csv");
    if(!abrirArchivo(&fpSimulacionR, nomArchSim, "wt", CON_MSJ))
    {
        return ERR_ABRIENDOARCH;
    }
    fprintf(fpSimulacion, "%s\n", primeraLinea);
    if(verPrimero(&lista, &infoAnt))
    {
        strcpy(reg.nomArch, infoAnt.nomArch);   /// inicializa informe resumido
        strcpy(reg.nivelDisimiliridad, infoAnt.info[0]);
        strcpy(reg.tDeCorte, infoAnt.info[1]);
        strcpy(reg.cantTopeIndiv, infoAnt.info[2]);
        reg.cantCorridas = 0;
        reg.cantEstimEspecies = 0;
        reg.tiempoEjecucion = 0.0;
        while(sacarPrimero(&lista, &info))
        {

            if(compararXNomArchDiversidEtc(&infoAnt, &info) != 0)
            {
                fprintf(fpSimulacionR,
                        "%s\n%s\n%s\n%s\n"
                        "Corridas: %d"
                        " - Cant. Estimada de especies: %I64d"
                        " - Tiempo total de ejecución: %ld\n\n",
                        reg.nomArch,
                        reg.nivelDisimiliridad,
                        reg.tDeCorte,
                        reg.cantTopeIndiv,
                        reg.cantCorridas,
                        reg.cantEstimEspecies / reg.cantCorridas,
                        reg.tiempoEjecucion);
                infoAnt = info;
                strcpy(reg.nomArch, infoAnt.nomArch);   /// inicializa informe resumido
                strcpy(reg.nivelDisimiliridad, infoAnt.info[0]);
                strcpy(reg.tDeCorte, infoAnt.info[1]);
                strcpy(reg.cantTopeIndiv, infoAnt.info[2]);
                reg.cantCorridas = 0;
                reg.cantEstimEspecies = 0;
                reg.tiempoEjecucion = 0.0;
            }
            long        cantClusters    = -1;
            long        tiempoEjecucion = -1;
            fprintf(fpSimulacion, "%s\n", info.nomArch);
            ciclo = 0;
            while(ciclo < 20 && strlen(info.info[ciclo]))
            {
                fprintf(fpSimulacion, "%s\n", info.info[ciclo]);
                ciclo++;
            }
            fprintf(fpSimulacion, "\n\n");
            sscanf(info.info[ciclo - 1],
                   "\t%ld\t%*d\t%*d\t%*f\t%ld",
                   &cantClusters,
                   &tiempoEjecucion);               /// contabiliza:
            reg.cantCorridas++;                         /// catCorridas
            reg.cantEstimEspecies += cantClusters;      /// cantClusters
            reg.tiempoEjecucion += tiempoEjecucion;     /// tiempoEjecucion
        }
        fprintf(fpSimulacionR,
                        "%s\n%s\n%s\n%s\n"
                        "Corridas: %d"
                        " - Cant. Estimada de especies: %I64d"
                        " - Tiempo total de ejecución: %ld\n\n",
                        reg.nomArch,
                        reg.nivelDisimiliridad,
                        reg.tDeCorte,
                        reg.cantTopeIndiv,
                        reg.cantCorridas,
                        reg.cantEstimEspecies / reg.cantCorridas,
                        reg.tiempoEjecucion);
    }
    fclose(fpSimulacion);
    return 0;
}
```

Código C de la aplicación GenerarInforme.exe: (main.c) - Volver

Código C de la aplicación GenerarInforme.exe: (main.h) - Volver

```
#ifndef MAIN_H_
#define MAIN_H_

#include <stdio.h>
#include <stdlib.h>


#include "lista.h"

#define ERR_CANTARGMAIN             1
#define ERR_PARAMMAIN               2
#define ERR_ABRIENDOARCH            4
#define ERR_ABRIENDOARCHTMP         8
#define ERR_LEYENDOLIST             16
#define ERR_PROCSIMULA              32
#define ERR_CONTANDOIND             64


//#define ARCH_ENT        ".\\S85.phy.cgi.fn.list"
#define ARCH_ENT        ".\\datosc\\S86.phy.cgi.fn.list"

#define ARCH_TEMP       ".\\tmp%02d.csv"
/* comenzando con el archivo tmp00.csv */

#define INFO_1          ".\\1merInfo.csv"
#define INFO_2          ".\\Info.csv"


#define CON_MSJ         1

#define abrirArchivo( X, Y, Z, Q )                                          \
            ( (*(X) = fopen(Y, Z)) == NULL  ?                               \
                  ( Q == CON_MSJ ?                                          \
                     !fprintf(stderr,                                       \
                             "ERROR - abriendo \"%s\" en modo \"%s\"\n",    \
                             Y, Z) | 0  :                                   \
                     0 ) :                                                  \
                  1 )

typedef struct
{
    char        nomArch[300],
                nivelDisimiliridad[100],
                tDeCorte[100],
                cantTopeIndiv[100];
    int         cantCorridas;
    long long   cantEstimEspecies;
    long        tiempoEjecucion;
} t_reg;
#endif
```

Código C de la aplicación GenerarInforme.exe: (lista.c) - Volver

```
#include "lista.h"

/**
 * funcion:
 * recibe:
 * devuelve:
 * precond.:
 * obser.:
 */
/**
 * funcion:    crearLista
 * recibe:     la lista
 * devuelve:   (n/a)
 * precond.:   (n/a)
 * obser.:     crea la lista como lista vacía
```

Código C de la aplicación GenerarInforme.exe: (lista.c) - Volver

```
 */
void crearLista(t_lista *p)
{
   *p = NULL;
}


/**
 * funcion:     insertarEnOrden
 * recibe:      la lista
 *              la info a cargar
 * devuelve:    TODO_BIEN si es exitosa
 *              SIN_MEM en caso de fracaso
 * precond.:    que la lista esté inicializada
 * obser.:      inserta con claves duplicadas
 */
int  insertarEnOrden(t_lista *p, const t_info *d)
{
    t_nodo *nue;
    while(*p && compararXNomArchDiversidEtc(d, &(*p)->info) >= 0)
        p = &(*p)->sig;

    nue = (t_nodo *)malloc(sizeof(t_nodo));
    if(nue == NULL)
        return SIN_MEM;
    nue->info = *d;
    nue->sig  = *p;
    *p        = nue;
    return TODO_BIEN;
}




int  sacarPrimero(t_lista *p, t_info *d)
{
    t_nodo *aux = *p;
    if(*p == NULL)
        return 0;
    *p = aux->sig;
    *d = aux->info;
    free(aux);
    return 1;
}



int  compararXNomArchDiversidEtc(const t_info *d1, const t_info *d2)
{
    int cmp = strcmpi(d1->nomArch, d2->nomArch);
    if(cmp)
        return cmp;
    if(strstr(d1->info[0], "uniq") && strstr(d2->info[0], "uniq"))
        return 0;
    if(strstr(d1->info[0], "uniq"))
        return -1;
    if(strstr(d2->info[0], "uniq"))
        return 1;
    return strcmpi(d1->info[0], d2->info[0]);
}



int  verPrimero(const t_lista *p, t_info *d)
{
    if(*p == NULL)
        return 0;
    *d = (*p)->info;
    return 1;
}
```

Código C de la aplicación GenerarInforme.exe: (lista.h) - Volver

```
#ifndef LISTA_H_
#define LISTA_H_

#include <stdlib.h>
#include <string.h>
#include <stdio.h>


#define CLA_DUP         2
#define TODO_BIEN       1
#define SIN_MEM         0

#define LISTA_VACIA     -1


typedef struct
{
   char           nomArch[300],
                  info[20][100];
} t_info;


typedef struct s_nodo
{
   t_info         info;
   struct s_nodo *sig;
} t_nodo, *t_lista;


void crearLista(t_lista *p);

int  insertarEnOrden(t_lista *p, const t_info *d);

int  sacarPrimero(t_lista *p, t_info *d);

int  verPrimero(const t_lista *p, t_info *d);

int  compararXNomArchDiversidEtc(const t_info *d1, const t_info *d2);


#endif
```

Volver

10.2 Artículos

INFERENCIA DE PARÁMETROS DE BIODIVERSIDAD POR MEDIO DE SIMULACIÓN

Cristóbal R. Santa María† y Marcelo A. Soria‡

†Departamento de Ingeniería e Investigaciones Tecnológicas, Universidad Nacional de La Matanza, Florencio Varela 1903, 1754 San Justo, Argentina, csanta_maria@ing.unlam.edu.ar, www.unlam.edu.ar
‡Cátedra de Microbiología, Facultad de Agronomía, Universidad de Buenos Aires. INBA-CONICET, Av. San Martín 4453, 1427 CABA, Argentina, soria@agro.uba.ar, www.agro.uba.ar

Resumen: En estudios metagenómicos sobre comunidades microbianas suele ocurrir que, pese a contar en términos absolutos con un gran número de datos, éstos resultan insuficientes para establecer estimaciones adecuadas de parámetros de biodiversidad. La subestimación de la riqueza de especies producida por las técnicas no paramétricas habitualmente utilizadas, motiva la construcción del Algoritmo de Recuento de Especies (ARE) que, sobre un modelo de simulación, realiza una expansión de la muestra y logra así mejorar la estimación. El procedimiento parte de la estimación de la probabilidad de especie "nueva" construida por Turing y agrega, en cada iteración, un individuo simulado a la muestra original, obteniendo un nuevo recuento de especies. Se aplica ARE sobre dos conjuntos muestrales y se comparan sus resultados con estrimaciones no paramétricas. También se prueba su desempeño sobre una comunidad simulada según la serie log de Fischer. Los resultados confirman la mejora producida en la estimación de riqueza.

Palabras claves: *riqueza, especie, estimación, simulación*

1. INTRODUCCIÓN

Para evaluar la biodiversidad de una comunidad es preciso determinar cuantas especies forman parte de la misma y en que proporción se hallan. Una alternativa utilizada es el análisis basado en el gen 16S rRNA, de alta conservación a lo largo del proceso evolutivo, que permite apreciar con exactitud las diferencias taxonómicas [1]. Una vez secuenciadas las cadenas de ADN del gen, obtenidas de una muestra de material biológico, cada secuencia representa a un individuo distinto. Las secuencias se alinean y filtran por procedimientos estándar y luego se miden sus "distancias genéticas" a fin de realizar un agrupamiento según el grado de similitud que revelen. Un umbral de disimilaridad entre el 3% y el 5% es propio de individuos de la misma especie que integrarán el mismo agrupamiento. Es decir, para contar cantidad de especies y averiguar su distribución en la muestra de material, habrá que contar "clusters" y cantidad de secuencias que los integran.

Pero, cuando se desea inferir desde una muestra la riqueza de todo el medio biológico se presentan dificultades de carácter estadístico que provienen de la gran cantidad de microorganismos que integran realmente la comunidad y de la existencia de una mayoría de especies que se encuentran en muy baja proporción y resultan entonces "raras". El tamaño de la comunidad y la rareza estadística se suman a limitaciones tecnológicas y/o económicas que impiden aumentar la cantidad de individuos muestreados, introduciendo así un grado de incertidumbre en las estimaciones de la riqueza comunitaria que no puede tratarse con las técnicas estadísticas habituales.

Existen distintos modelos a partir de los cuales es posible abordar esta situación, pero sus resultados suelen subestimar la real cantidad de taxones presentes en la comunidad [2] Un enfoque alternativo a la estimación no paramétrica de la riqueza, es el que aportan las curvas de rarefacción [3] El método permite estimar la riqueza de un medio al aplicar una técnica de remuestreo planteada por Efron en [4]. Consiste en ir contando la cantidad de especies diferentes, tomando cada vez un orden distinto y aleatorio de los individuos de la muestra y estableciendo el número acumulado promedio para cada cantidad i de individuos examinados. La curva resultante tiene un aspecto suave, creciente y asintótico con el valor de riqueza de la comunidad en la medida en que crece el tamaño muestral.

La idea del Algoritmo de Recuento de Especies (ARE) parte de dos conceptos. Por un lado se consideró la existencia de una estimación de la probabilidad de hallar una especie nueva cuando se selecciona un nuevo individuo para integrar la muestra. Por otro, se observó el hecho de que la curva de rarefacción debe alcanzar un comportamiento asintótico horizontal para un tamaño muestral suficientemente grande, por lo cual una curva de acumulación de especies distintas debiera observar un comportamiento similar cuando aumenta la cantidad de individuos en la muestra.

2. EL MODELO EXPERIMENTAL

Al seleccionar aleatoriamente *n* individuos de una comunidad que contiene un número finito de especies *S*, las cantidades $n_1, n_2, \ldots, n_r, \ldots, n_k$ son, en cada caso, el número de especies que

contabilizan *r* individuos entre los *n* seleccionados. De tal forma $\sum_{r=1}^{k} r n_r = n$ Es claro que n_0 expresa la cantidad de especies que no aparece representada por ninguno de los *n* individuos elegidos y que la cantidad de especies en la comunidad es $S = \sum_{r=0}^{k} n_r$ Si se denomina S_n a la cantidad de especies contabilizadas al tomar *n* individuos resulta $S_n = S - n_0$. Hay que observar que al agregar un nuevo individuo a una muestra, éste puede resultar perteneciente a una especie ya presente en la muestra, o a una nueva aún no contabilizada. Para formalizar se define:

Definición 1: *Dada una muestra de tamaño n sea, para cada i, con* $i = 1,2,3,\ldots$*, la variable aleatoria* S_i *que toma los valores* $S_i = S_{i-1}$ *y* $S_i = S_{i-1} + 1$ *con probabilidades respectivas* $1 - p_i$ *y* p_i *siendo además* $S_0 = S_n$*. La sucesión de variables aleatorias* $S_1, S_2, S_3, \ldots$*se denomina en adelante Proceso Aleatorio de Cantidad de Especies.*

Nota 1: *La interpretación que constituye el modelo experimental identifica a* S_i *como la cantidad de especies distintas presentes en una muestra de tamaño n+i. Además* p_i *se interpreta como la probabilidad de que al incorporar un nuevo individuo a una muestra de tamaño n+* $i-1$*, éste corresponda a una especie nueva no presente hasta ahora en la muestra.*

El siguiente paso es construir una estimación de la probabilidad p_i . En [5] Good prueba que la probabilidad de que, elegidos *n* individuos, al seleccionar uno nuevo éste resulte de una especie hasta ahora no contabilizada, puede aproximarse por el cociente $T = \frac{n_1}{n}$ dónde n_1 es el número de especies que aparece una vez en la muestra elegida y que debe suponerse mayor estricto que 0. Esta idea, que se atribuye a Turing, es la que aquí se utiliza para estimar p_i . Se propone entonces $T_i = \frac{n°sgletones}{n+i-1}$ como fórmula de cálculo de la probabilidad de especie nueva asociada al proceso aleatorio de cantidad de especies. El número de singletones en cada muestra de tamaño $n+i-1$ refiere a los clusters formados por un solo individuo, cuando se realiza el procedimiento de agrupamiento de secuencias indicado en la introducción y la consiguiente identificación de cada cluster con una especie distinta.

Teorema1 : $\lim_{i\to\infty} T_i = 0$

Prueba. Se denota por $n_{r(i-1)}$ al número de especies que aparecen *r* veces en la muestra de tamaño $n+i-1$. Claramente $n_{r(i-1)} \geq 0$ para todo $i = 1,2,\ldots$con $r = 0,1,\ldots,k_i$ La cantidad $n_{0(i-1)}$ es el número de especies no presentes en la muestra y $n_{1(i-1)}$ el número de especies que aparecen una sola vez, es decir la cantidad de singletones. También es claro que $S = \sum_{r=0}^{k_i} n_{r(i-1)}$ para todo *i*. Resulta así que $0 \leq n_{1(i-1)} \leq S$. Por lo tanto se cumple que $0 \leq \frac{n_{1(i-1)}}{n+i-1} \leq \frac{S}{n+i-1}$ dónde *n* es el tamaño inicial de la muestra. Al pasar al límite en cada miembro de la desigualdad se tiene $\lim_{i\to\infty} \frac{n_{1(i-1)}}{n+i-1} = \lim_{i\to\infty} T_i = 0$ □

Teorema 2: $\lim_{i\to\infty} S_i \leq S$

Prueba. Por construcción $S_{i-1} \leq S_i$ para todo *i*. Resulta entonces que la sucesión de variables aleatorias es monótona creciente. Además para cada *i* se cumple que $S_i = \sum_{r=1}^{k_i} n_{ri} \leq n_{0i} + \sum_{r=1}^{k_i} n_{ri} = S$. Como

$n_{0i} \geq 0$ se tiene $S_i \leq S$ para todo *i*. Por lo tanto $\lim_{i\to\infty} S_i \leq S$

□

Nota 2: *El* **Teorema 2** *muestra que la sucesión de los* S_i *es convergente a un límite en forma asintótica. Además ese límite no superará la cantidad S de especies presentes en la población.*

3. EL ALGORITMO DE RECUENTO DE ESPECIES

El algoritmo realiza la simulación por la técnica de Monte Carlo. Dado el tamaño $n+i-1$ de la muestra se determina el valor del estimador de la probabilidad de especie nueva. Ese valor permite constituir los intervalos $[0, T_i]$ y $(T_i, 1]$ de modo que al elegir un número aleatorio r tal que $0 \leq r \leq 1$, si cae dentro del primer intervalo el nuevo individuo simulado corresponda a una especie nueva y si cae dentro del segundo intervalo es un ejemplar de una especie conocida. Si ocurre lo primero, la cantidad de especies en el medio se incrementa en 1 y si no, se utilizan las proporciones de cada especie, existentes en la muestra, para asignar por medio de un nuevo número aleatorio la especie ya conocida, a la cual pertenece el nuevo individuo. Así se van agregando individuos hasta que el cálculo se detiene cuando el valor de T_i alcanza una cantidad suficientemente pequeña prefijada. Para efectuar pruebas el algoritmo se ha programado utilizando el software libre R [6]

Las pruebas realizadas sobre los conjuntos de muestras SRX008158 y ERR009564 extraídos de la base de datos de NCBI [7] arrojaron estimaciones superiores a las obtenidas por los estimadores no paramétricos de uso habitual CHAO y ACE [8]. En este caso se utilizaron como valores de corte de la simulación $T_i = 0.03$ y $T_i = 0.015$ respectivamente, que se consideraron umbrales no muy exigentes. Los resultados comparados se exhiben en la Figura 1.

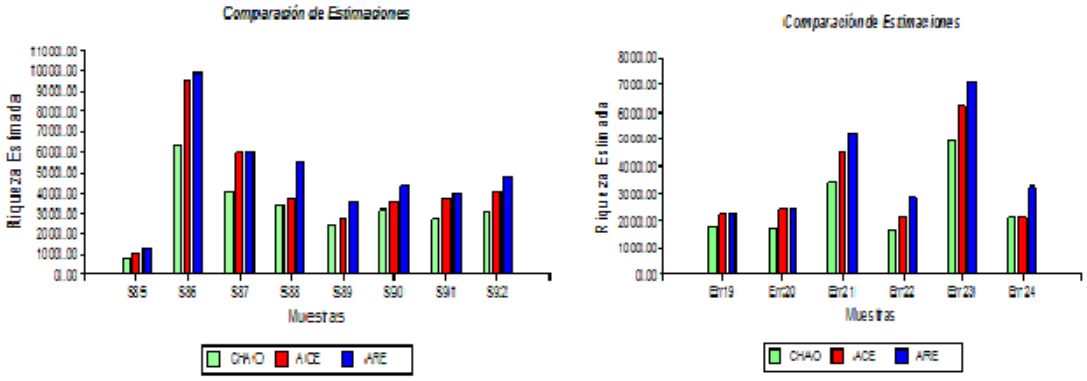


Figura 1: Comparación de estimaciones

4. PRUEBAS SOBRE UNA COMUNIDAD SIMULADA

Por causas tecnológicas y económicas no es posible contar con secuencias del gen 16S rRNA correspondientes a todos los individuos de una comunidad microbiana real. Por esta razón, para realizar una prueba sobre una población completa, se construyó una simulación de la misma. Se utilizó para ello la serie log analizada por Fischer en [9] $\alpha x, \frac{\alpha x^2}{2}, \frac{\alpha x^3}{3}, \ldots, \frac{\alpha x^m}{m}$ que modela la cantidad esperada de especies que están representadas por 1, 2,…, *m* individuos en la población. Como se prueba en [9] si *N* es el tamaño de la población y *S* la cantidad de especies en ella contenida, se cumplen las relaciones $\frac{S}{N} = [(1-x)/x][-\ln(1-x)]$ y $\alpha = \frac{N(1-x)}{x}$ De acuerdo a esto pueden calcularse valores de *S* y *N* a partir de cantidades α y *x* cuya significación ecológica desarrolla Magurran en [10]. Se utilizaron entonces los valores $\alpha = 5000$ y x=0.995 sugeridos en [10] para obtener una comunidad integrada por 898341 individuos distribuidos entre 26332 especies.

La prueba experimental consistió en aplicar el algoritmo ARE, sobre una muestra inicial de 1000 individuos, realizando distintas cantidades de iteraciones. En cada corrida se obtuvo también el valor que alcanzó la probabilidad de especie nueva T_i al cortar la simulación. La Tabla 1 exhibe los resultados.

ARE/60000 Iteraciones 14615	ARE/200000 Iteraciones 19271	ARE/500000 Iteraciones 24559	ARE/897541 Iteraciones 25327
$T_{60000} = 0.091$	$T_{200000} = 0.026$	$T_{500000} = 0.011$	$T_{897541} = 0.005$

Tabla1: Estimación ARE de la riqueza comunitaria según cantidad de iteraciones

La Tabla 2 evidencia las mejoras que ARE produce en la estimación de riqueza conforme aumenta el número de iteraciones.

Real	CHAO	ACE	ARE/45000	ARE/60000	ARE/200000	ARE/500000
26332	6699	6751	12821	14615	19271	24559
100%	25%	26%	49%	56%	73%	93%

Tabla 2: Comparación del desempeño de estimadores de riqueza

5. CONCLUSIONES

El Algoritmo de Recuento de Especies presentado produce una mejora sensible en las estimaciones de riqueza comunitaria. Se plantea el desafío de optimizar sus tiempos de ejecución computacional a fin de posibilitar su aplicación, de forma estándar, en las evaluaciones de biodiversidad.

6. REFERENCIAS

[1] N. YOUSSEF y M. ELSHAHED *Species richness in soil bacterial communities: A proposed approach to overcome sample size bias*. Journal of Microbiological Methods, 75 (2008), pp. 86-91

[2] J. HUGHES, J. HELLMAN, T. RICKETTS y B. BOHANNAN *Counting the uncountable: statistical approaches to estimating microbial diversity*. Appied and Environmental Microbiology, (2001), pp. 4399-4406.

[3] J. HUGHES y J. HELLMAN *The Application of Rarefaction Techniques to Molecular Inventories of Microbial Diversity*. Methods in Enzymology. Vol 397 (2005).

[4] B. EFRON *Computers and theory of statistics: thinking the unthinkable*. Technical Report N° 39. Division of Biostatistics. Stanford University (1978)

[5] I. J. GOOD *The Population Frequencies of Species and Estimation of Population Parameters*. Biometrika. Vol 40 N° ¾ (1953), pp. 237-264

[6] http://www.r-project.org/

[7] http://www.ncbi.nlm.nih.gov/

[8] A. CHAO y S. LEE *Estimating the Number of Classes via Sample Coverage*. Journal of American Statistical Association. Vol 01. 87, N° 417. (1992)

[9] R. FISCHER, S. CORBETT y C. WILLIAMS. *The Relation Between the Number of Species and the Number of Individuals in a Random Sample of an Animal Population*. The Journal of Animal Ecology. British Ecological Society. Vol 12 N°1 (1943)

[10] A. E. MAGURRAN. *Measuring Biological Diversity.* Blackwell Science Ltd Access published February 3. The Oxford University Press. 2004

Simulation applied to the estimation of microbial richness

Cristóbal Santa María[1], Marcelo Soria[2]
[1]UNLAM.San Justo. Argentina
[2]FAUBA. Buenos Aires. Argentina

Introduction

As part of our research on applications of data mining to metagenomic surveys we developed the ARE procedure to assess the richness of a microbial community [1]. For this task we used as a marker the sequence of the gene coding for the 16S ribosomal RNA (16S rRNA). In previous reports we presented computational proofs of the convergence of the ARE algorithm with different sampling sets and with simulated communities. In this opportunity we present some mathematical results regarding convergence and a case study of richness estimation with a real community. The estimates obtained with these evaluations confirm the higher accuracy of the inferred values compared to the commonly used non-parametric estimators CHAO and ACE, which had also been observed on simulated community with a Fisher distribution [1],[2].

Materials and Methods

A progressive experimental model was developed starting with a sample and increasing its size one individual at a time. At every step, or state, we assume a certain probability that the next individual to be added belonged to a "species" not accounted for in the community up to that step. If that were the case, the count of species in the community is increased by one, else the number of species remains the same and the individual is assigned randomly to one of the existing species, which gets its count of individuals increased by one. In both cases, the total number of individuals changes and the system reaches a new state. At every state it is possible to estimate the species richness of the community. Initially the richness increases with every successive state, but after n individuals were added to the system, the richness does not change anymore, or shows a very small relative change. In detail, to estimate the probability that the individual added to the community at step i belongs to a new species we use the quotient:

T_i = n° singletons / (i-1)

The simulation is a Monte Carlo procedure that starts with a sample of the initial community, for example a survey of amplicons obtained by next generation sequencing, and generates the complete community, on which the richness is calculated. In turn, this is an estimate of the richness in the real population. The stochastic procedure can be formalized as follows:

Definition 1: given a sample of size n be S_i, for each i=1,2,…, a random variable representing the number of species that takes values $S_i=S_{i-1}$ and $S_i = S_{i-1}+1$ with probabilities $1-p_i$ and p_i respectively, with $S_0=S_n$. The succession of random variables S_1, S_2, S_3… is designated "Random Process of the Number of Species".

Results

Two properties of the process are proved:

Property 1: If $i->\infty$ $T_i->0$

Property 2: If $i->\infty$ $S_i->0$ where S is the real number of species in the community.

Both properties, under the assumption of the adequate estimate that yields T_i, warrant the convergence of the ARE procedure to a value that is less than or equal to the real community richness [2].

To perfom a test on real data we selected a sample of 16316 sequences of the 16S rRNA gene described in [3] and designated FS396.archaea. The sample was considered to describe the full community because its rarefaction curve reaches an asymptote. Using the MOTHUR suite we filtered, aligned and cluster the sequences, and determined it contained 346 OTUs or species. A random sample of 1600 was withdrawn and six runs of the ARE procedure were ran with it. The results are shown in the table below.

Table 1

Run Number	Number of estimated OTUs	Simulated individuals
1	325	1680001
2	253	1245882
3	305	1680001
4	374	2494888
5	227	1440045
6	401	2111128

Average number of ARE estimated OTUs =314
95% Confidence interval: (265,363)
Chao estimate=163 ACE estimate = 269

Conclusions

The results obtained from a simple drawn from a real community confirmed the good performance of ARE, in agreement with the observations previously reported from simulated communites, In both situations ACE only reached the lower limit of the confidence interval for ARE.

References

1. Santa Maria C, Soria M: **Estimation of Species Richness in Microbial Communities.** Memorias del 3er Congreso Argentino de Bioinformática y Biología Computacional 2012, 74

2. Santa María C, Soria M: **Inferencia de Parámetros de Biodiversidad por medio de Simulación.** MACI 2013, **Vol 4**: 5-8

3. Huber JA, Welch DBM, Morrison HG, Huse SM, Neal PR, Butterfield DA et al: **Microbial population structures in the deep marine biosphere.** Science 2007, **318**: 97–100.

4. Haegeman B, Hamelin J, Motitarty J, Neal P, Dushoff J, Weitz J: **Robust estimation of microbial diversity in theory and in practice.** ISME Journal 2013, 1-10

DATA MINING EN EVALUACIONES DE BIODIVERSIDAD

Luis López. Departamento de Ingeniería. UNLaM
Pablo Martínez. Departamento de Ingeniería. UNLaM
Ariel Cacho Mendoza. Departamento de Ingeniería. UNLaM
Marcelo Soria. Facultad de Agronomía, Cátedra de Microbiología UBA
Cristóbal R. Santa María. Departamento de Ingeniería. UNLaM
Florencio Varela 1903 San Justo Pcia. de Buenos Aires
54-011-44808952
llopez@ing.unlam.edu.ar
pablowmartinez@yahoo.com.ar
arielcm@gmail.com
soria@agro.uba.ar
csanta_maria@ing.unlam.edu.ar

RESUMEN

Las modernas técnicas de secuenciación de ADN transforman su estructura química en secuencias informáticas de símbolos cada una de las cuales puede ser vista como una instancia de una base de datos. Es posible entonces aplicar técnicas para clasificar casos y predecir patrones de comportamiento de forma similar a como se lo hace sobre otros dominios como las finanzas, el marketing o el texto, aunque la complejidad del dominio microbiológico pueda llevar a una tarea un poco más ardua. En tal sentido la aplicación de data mining en los estudios genómicos es un hecho consolidado en la investigación biológica pues en ella también se trata de clasificar y descubrir patrones sobre grandes bases de datos con el auxilio de técnicas combinadas de aprendizaje automático, estadística y visualización lo que en suma no es más que la definición ontológica de la minería de datos.

El trabajo aquí presentado se refiere a secuencias de ADN correspondientes a distintos microorganismos extraídas de muestras de suelo con el objetivo de evaluar los patrones de riqueza y diversidad de la comunidad microbiológica que lo integra.

En particular cada secuencia de ADN correspondiente al gen 16S rRNA que integra la muestra se identificará con un organismo distinto. La tecnología de secuenciación actual es capaz de obtener miles de estas cadenas de símbolos correspondientes a los cuatro componentes básicos del ADN: A-adenina, T-timina, C-citocina y G-guanina. Cada parte de un gen será entonces una secuencia de unos cientos de estos símbolos colocados en algún orden. Tal como se hace por ejemplo en text mining, se puede definir una distancia conveniente entre secuencias y con ella producir un clustering que agrupe las secuencias según su similitud. Así, eligiendo un umbral de disimilaridad adecuado, cada agrupamiento estará integrado por secuencias correspondientes a individuos de la misma especie, Estos clusters se denominan Unidades Taxonómicas Operacionales y a partir de su distribución de abundancia en la muestra, se pretende establecer el patrón de riqueza de la comunidad, lo que significa establecer el número de especies que hay en la misma. Esta tarea se topa con un serio problema estadístico pues en microbiología más del 70% de las especies pueden ser estadísticamente raras a la vez que un 10% es muy abundante. De tal forma las muestras no contienen individuos de muchas especies presentes y a su vez presentan muchos individuos de las especies dominantes. Es decir; toda muestra resulta pequeña para una inferencia estadística simple de la riqueza poblacional. El algoritmo de recuento de especies ARE, ya presentado en otros trabajos (1) y (2), mejora las estimaciones no paramétricas habituales y las hace compatibles con las apreciaciones ecológicas. En términos más generales el algoritmo resuelve en forma eficiente el problema de inferir desde una muestra de casos el número de clases de casos que hay en una población que contiene una alta proporción de clases raras. Este problema se reconoce también, por ejemplo, en el análisis de texto donde cada palabra distinta es una clase y hay palabras muy poco frecuentes (3). Hay que remarcar que el número inferido para la riqueza como cantidad de especies distintas, o si se quiere palabras distintas, constituye una guía imprescindible para afinar el clustering que se realice sobre nuevas muestras de la población para determinar una clasificación estable y aplicable luego para predicción.

En este trabajo se planteó el objetivo de desarrollar un programa escrito en Lenguaje C o C++ que permitiera reemplazar al programa del algoritmo ARE escrito en lenguaje R con el fin de mejorar los tiempos de ejecución Se estudian además las posibilidades de paralelización en la ejecución de los algoritmos.

CONTEXTO

La línea de trabajo que aquí se presenta se inscribe en el proyecto de investigación de técnicas de minería de datos aplicadas sobre bases de secuencias de ADN correspondientes a una colección de microorganismos. El proyecto tiene la finalidad de buscar instrumentos adecuados para evaluar la biodiversidad. A su vez se intenta desarrollar una programación que mejore los tiempos de procesamiento con respecto a los de ejecución en lenguaje R que fue utilizado para experimentación inicial.

INTRODUCCIÓN

Para abordar el problema planteado se desarrolló un modelo experimental a partir de la riqueza de una muestra. En ese estado inicial se considera la probabilidad de que al elegir un próximo individuo para incorporar a la muestra, éste resulte de una especie hasta ahora no contabilizada. Si tal cosa ocurre, se suma una nueva especie al total contado. Si no, el número de especies queda igual. En cualquier caso ha variado el número de individuos considerados y el sistema se halla en un nuevo estado. Cuando se ha elegido la cantidad suficiente de individuos se obtiene una estimación de la riqueza medida en este caso en número de especies. Para estimar la probabilidad de que el próximo individuo que se incorpore a la muestra sea de una especie nueva se usa, en el paso *i* del algoritmo, el cociente:

$$\hat{T}_i = \frac{n°sgletones}{i-1}$$

Se realiza entonces una simulación por la técnica de Monte Carlo que a partir de la muestra inicial de tamaño n genera una comunidad simulada cuya cantidad de especies estima la riqueza de la comunidad real. (1)

El proceso estocástico modelado se formaliza:

Definición 1: Dada una muestra de tamaño n sea, para cada i, con $i = 1,2,...$, la variable aleatoria S_i que toma los valores $S_i = S_{i-1}$ y $S_i = S_{i-1} + 1$ con probabilidades respectivas $1 - p_i$ y p_i siendo además $S_0 = S_n$. La sucesión de variables aleatorias $S_1, S_2, S_3, ...$ se denomina en adelante Proceso Aleatorio de Cantidad de Especies.

Se prueban dos propiedades de tal proceso :

Propiedad 1: $\lim_{i \to \infty} T_i = 0$

Propiedad 2: $\lim_{i \to \infty} S_i \leq S$ donde *S* es el número real de especies en la comunidad. Ambas propiedades, bajo el supuesto de la adecuada estimación de p_i que proporciona $\hat{T}_i$, aseguran la convergencia del procedimiento ARE a un valor menor o igual que el de la real riqueza poblacional. (2)

Los pasos del algoritmo son (4):

v- Dada la muestra elegida, de tamaño n , y su agrupamiento en OTUs, se determina el valor inicial del estimador de Turing $\hat{T}_{i+1} = \frac{n°sgletons}{i}$ siendo $i = n$

vi- Se elige un número aleatorio r , tal que $0 \leq r \leq 1$ y se pregunta si está en el intervalo $[0, \hat{T}_{i+1})$. Si es así, se realiza $S_{i+1} = S_i + 1$ y se va al paso 4. Si ocurre lo contrario se realiza $S_{i+1} = S_i$ y se va al paso 3

vii- Se utiliza la distribución de abundancia de la muestra para calcular la proporción de individuos que están en OTUs de $1,2,...,n$ individuos y con estas proporciones se determina, por un sorteo de acuerdo a ellas, a que grupo de OTUs ya conocidas pertenece el nuevo individuo. Para establecer a que OTU específica, de entre las de este grupo, corresponde el nuevo individuo se realiza un nuevo sorteo con probabilidad uniforme para cada OTU del grupo.

viii- Sea el nuevo individuo de una nueva especie o no, la muestra tiene ahora un elemento más. Se pregunta entonces si el procedimiento debe cortarse porque se cumple el criterio elegido para ello en cuyo caso la simulación ha finalizado. Si el criterio de corte no se cumple, se asigna entonces $i \leftarrow i + 1$, se calcula la nueva distribución de abundancia y la nueva estimación de Turing y se repite desde el paso 2.

En cuanto a las muestras del gen 16s rRNA que caracterizan a cada individuo estas fueron extraídas del repositorio internacional NCBI (5). Su procesamiento inicial se realizó por medio del software libre MOTHUR (6) para realizar el clustering con un umbral de disimilaridad del 5% suficiente para reconocer individuos de la misma especie. La salida de tales procesos es un archivo ".list".

LÍNEAS DE INVESTIGACIÓN Y DESARROLLO

El presente trabajo se considera la etapa de desarrollo tecnológico de la línea de investigación que procura establecer estimaciones de patrones de riqueza y diversidad a partir de relevamientos denominados metagenómicos pues reúnen partes de genomas de muchos individuos de comunidades microbianas.

Al reemplazar los programas experimentales escritos en lenguaje R, se plantea entonces como estrategia resolver el problema en las siguientes etapas:

- Proceder a la lectura del archivo ".list" generado por MOTHUR
- Elegir un generador de números al azar
- Generar la simulación de la comunidad evaluando en ella la cantidad de especies.

El objetivo es reducir los tiempos de ejecución del algoritmo preparándolo para su incorporación a una plataforma de procesos estándar de secuencias genómicas.

RESULTADOS Y OBJETIVOS

a) Lectura del archivo ".list".

Se trata de un archivo de texto en formato CSV, en el mismo se utiliza el carácter de tabulación como separador de campo, y dentro de cada campo, si corresponde, los distintos individuos de un cluster u OTU se separan con el carácter ',' (coma). La marca de fin de registro es la habitual (marca de fin de línea de texto 0x0D 0x0A). En el primer campo se indican los niveles de agrupamiento (unique, 0.00, 0.01, etc.), o sea el porcentaje de diferencia en las secuencias de ADN, En el segundo campo se indica el número de clusters (OTUs) de la muestra. A continuación los distintos clusters que pueden estar formados por uno o más individuos. La estrategia elegida, para su uso a futuro, es la de leer como texto el archivo ".list" y generar tantos archivos temporarios como registros tiene el archivo ".list" en el que se reemplazan los caracteres de tabulación por la marca de fin de registro además de reemplazar los caracteres ',' por tabulación. A continuación se leen los distintos archivos temporarios y se genera el informe "info.csv". Se han validado los resultados obtenidos. Se opta por la futura representación de

solo los seis primeros dígitos decimales a pesar de que se los calcula con precisión double.

b) Generador de números al azar.

Se analizaron distintos algoritmos de los existentes, y se decidió utilizar el algoritmo denominado ran3 adaptado de Knuth (7). Se ha probado el mismo generado un archivo de texto con 2.000.000.000 de números al azar demorando poco más de una hora, cuyo costo de ejecución en más del 99% se debe a la grabación del archivo.

Se generan 200.000.000 de números al azar en una matriz en que los tres primeros dígitos decimales de cada número al azar direccionan el número de fila y los tres siguientes el número de columna (truncando los restantes dígitos), totalizando en cada posición de la matriz la cantidad de veces que aparece cada uno de los números al azar. Para visualizar los resultados se generó el archivo "azar.csv" a partir del que se generan las planillas de cálculo "azar.ods" y "azar.xls" en las que se hace uso de distintas fórmulas con el objeto de visualizar el resultado obtenido. Observación: dado que la matriz de 1.000.000 de enteros excede el espacio de memoria que permite Windows, se ha hecho uso de un archivo de paginación programado ad hoc.

Esta prueba ha dado un detalle relacionado con el hecho de que, cuando el individuo generado corresponde a una especie preexistente, se deben generar uno o dos nuevos números al azar. Al ejecutar el programa generando 600.000.000 de números al azar y tomar uno de cada tres de ellos para la matriz descripta, se afecta en gran medida la uniformidad de la distribución.

Esto ha llevado a modificar la función ran3 para poder generar tres secuencias separadas con el mismo algoritmo.

c) Generar la simulación en que se toma como entrada el archivo ".list" y se procesa una de las "filas" del mismo.

Se aprovecha parte del programa del paso a), y para la carga de la información en lugar del uso de un vector se utiliza una lista simplemente enlazada (tras probar con listas doblemente enlazadas cuyo empleo no se justifica). El tipo de dato lista contiene la información de los totales en tanto que los nodos están ordenados por la cantidad de individuos en el cluster. La salida del programa está hecha por pantalla y se muestra cada 1000 individuos generados la información similar a la generada en el paso a). La salida por pantalla puede hacerse a un archivo de texto con ínfimas modificaciones.

Las modificaciones hechas sobre la rutina de inicialización y generación de números al azar permitirán poder inicializar más de un generador.

Se han contrastado los resultados obtenidos, y coinciden con lo esperable.

Se han utilizado las fuentes de información disponibles. Entre ellas el libro electrónico Numerical Recipes In C (8), los trabajos de Knuth (7) y el de Marsaglia (9)

Actualmente se trabaja en las posibilidades de paralelización del algoritmo que parecen bastante bajas, pero no se descarta que en tanto se actualiza la inserción de un nuevo individuo y el recálculo de frecuencias se pueda realizar otro hilo de ejecución con la generación del los nuevos números al azar para el nuevo individuo. En cambio parece posible paralelizar la corrida de varias simulaciones sobre el archivo de la muestra inicial.

Los programas se han desarrollado en una plataforma Windows XP, utilizado el entorno de desarrollo Code::Blocks con el compilador MinGW/gcc-4.7.1 de distribución gratuita. Los fuentes podrán ser compilados con ínfimas modificaciones, si las hubiera, en otras plataformas.

BIBLIOGRAFÍA

1- Santa María C. y Soria M. (2013) Simulation applied to the estimation of microbial richness Resumen 4CAB2C 14

2- Santa María C. y Soria M. (2013) Inferencia de Parámetros de Biodiversidad por medio de Simulación" MACI Vol 4 1: 5-8

3- Nádas, A. (1985) On Turing's Formula for Word Probabilities. IEEE Transactions on Acoustics, Speech and Signal Processing. Vol ASSP-33 N° 6.

4- Santa María C. y Soria M. (2011) Estimación de Biodiversidad por Data Mining y Simulación" XVII Congreso Argentino de Ciencias de la Computación CACIC2011. 969-978

5- http://www.ncbi.nlm.nih.gov/

6- www.mothur.org

7- Donald Knuth. "The Art of Computer Programming" http://www-cs-faculty.stanford.edu/~uno/taocp.html

8- William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery "Numerical Recipes in c" http://www2.units.it/ipl/students_area/imm2/files/Numerical_Recipes.pdf

9- Marsaglia G. "The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness" http://www.stat.fsu.edu/pub/diehard/

10- Marsaglia G. "Random Number Generator" http://www.cs.pitt.edu/~kirk/cs1501/animations/Random.html

11- Marsaglia G. "The Monty Python method for generating random variables" http://portal.acm.org/citation.cfm?id=292395.292453

12- Marsaglia G. "The Ziggurat Method for Generating Random Variables" http://www.jstatsoft.org/v05/i08/paper

Bioinformatics - Account Created in ScholarOne Manuscripts

onbehalfof+bioinformatics.editorialoffice+oup.com@manuscriptcentral.com
en nombre de
bioinformatics.editorialoffice@oup.com
jue 22/01/2015 05:14 p.m.
Para:

...
22-Jan-2015

Dear Prof. Santa Maria:

A manuscript titled Evaluation of richness in microbial communities (BIOINF-2015-0085) has been submitted by Prof. Cristobal Santa Maria to the Bioinformatics.

You are listed as a co-author for this manuscript. The online peer-review system, ScholarOne Manuscripts, automatically creates a user account for you. Your USER ID and PASSWORD for your account is as follows:

Site URL: https://mc.manuscriptcentral.com/bioinformatics
USER ID: csanta_maria@ing.unlam.edu.ar
To enter your account, please do the following:

1. Go to: https://mc.manuscriptcentral.com/bioinformatics
2. Log in using this information:

Your USER ID is csanta_maria@ing.unlam.edu.ar

You can use the above USER ID and PASSWORD to log in to the site and check the status of papers you have authored/co-authored. This password is case-sensitive and temporary. Please log in to https://mc.manuscriptcentral.com/bioinformatics to update your account information and change your password.

Thank you for your participation.

Sincerely,
Bioinformatics

Genetic and population analysis

Evaluation of richness in microbial communities

Cristóbal Santa María[1] and Marcelo Soria[2,*]

[1]Departamento de Ingeniería e Investigaciones Tecnológicas, Universidad Nacional de La Matanza. 1754 San Justo. Argentina.

[2]Cátedra de Microbiología Agrícola. Fac. Agronomía. Universidad de Buenos Aires. 1417 Buenos Aires. Argentina

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

ABSTRACT

Motivation: The richness of a microbial community is an important parameter to compare its structure with those of other communities across time and environments. The estimation of richness based on marker gene data obtained through Next Generation Sequencing faces several statistical problems. The current estimators, which are mostly derived for the analysis of macro-organisms, tend to grossly underestimate the richness of microbial communities. We developed a stochastic process to understand the effect of the structure of the population on the traditional richness estimators and introduce a new measure, the Algorithm for Quantifying Species (AQS)

Results: The AQS is a non-parametric estimator based on simulation that in our tests outperformed the traditional richness estimators, especially in the case of samples that are a very small fraction of the population and a large number of rare species, which is the frequent situation in metagenomics,

Contact: csanta_maria@ing.unlam.edu.ar soria@agro.uba.ar.

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 INTRODUCTION

The quantification of the biodiversity in microbial communities is a complex task since, in addition to the statistical problems that arise similar to those found in the inference of richness or diversity in other kind of populations (Magurran, 2011), biases also emerge due to the process of metagenomic DNA, from which different individuals are identified, (Schloss, 2010; Youssef and Elshahed, 2008; Huse et al., 2010). This work is limited to explore an experimental methodology to treat the statistical inference of the population richness, and it supposes solved or at least mitigated by appropriate precautions, the effects of the sequencing, alignment, filtering and any other process of DNA obtained from a community sample.

*To whom correspondence should be addressed.

The technique used for microbial richness measurements requires a marker gene, highly conserved through evolution (Schloss-Handelsman 2006). Thus, when the sequences belonging to individuals of the same sample reveal a certain percentage of variations between them, such variations can be attributed to the difference in species and, in general, in taxa, and not simply to random occurrences.

The sample once transformed into a set of marker gene sequences, it can be inferred through it, the richness of the community and perhaps also its distribution. This is where a significant statistical problem appears, since usually the sample size is insufficient for the task and, in the facts the richness is underestimated (Hughes et al., 2001). This is due to the presence of a significant proportion of rare species or taxa, in statistical terms, making it very unlikely to find in the sample individuals belonging to all or nearly all of them. Individuals of rare species are very few in relation to individuals of abundant species in the community and also rare species occur in greater numbers than the abundant species, so the sample size should be very large to infer a reasonably approximate value of the richness. In other words, the rarity and distribution of species seriously complicate the estimation of the population richness (Roesch, L. et al. 2007). This problem has been attempted to address by building parametric estimators as CHAO (Chao 1984) and ACE (Chao-Lee 1992) that even though they have improved the estimations they have not solved the inference, at least when it comes to microbial populations.

One line of current work (Haegeman et al., 2013) considers that the appropriate biodiversity assessment requires the analysis of a set of Hill indices that represent richness, entropy, etc. (Hill 1973) (O´Hara, R. 2005). This perspective is expanded with the development of extrapolating rarefaction curves to assess the richness (Chao et al. 2014). The present work proposes an alternative procedure to those mentioned by building a random process that is increasing the sample size in a simulated form, by using an estimation of the probability that, given a sample of size n, an upcoming individual who is

added to the sample corresponds to a new species. This estimation was reported by Alan Turing in 1941 (Good 1953), (Nadas 1985).

Thus the Algorithm for Quantifying Species (AQS) designed updates the number of individuals and species in each iteration and grows the number of species simultaneously with the decrease to zero of the probability of finding a new species. Although the resulting distribution statistically may not correspond with the actual of the population, tests from samples of simulated and real populations allow seeing that, with a previous processing of the sequences to mitigate any bias, richness is better appreciated. This method of experimental modeling of the community can be computationally feasible and reasonable by optimizing execution times and with processing capabilities in parallel.

2 MATERIALS AND METHODS

1- In metagenomics, properties of a community are analyzed from the genomes of the individuals who composed it. In particular, given a microbial community, it is possible to sample and process the DNA to obtain sequences of the chosen marker gene that, in the present work, is the 16S rRNA (Armougom and Raoult 2009). Each sequence of this gene then corresponds to a distinct individual that integrates the sample and it is possible to gather sequences according to proximity criteria evaluated by the model of genetic distance of Jukes-Cantor or another similar (Hillis et al. 1996). Then, a threshold of dissimilarity between sequences can be taken, from which they will be considered as different species (or generally taxa). So different clusters are created that represent each one a species (or taxon) composed of individuals within the sample and are similar according to the threshold chosen (Schloss-Handelsman 2005). The *n* original sequences of the sample are distributed in clusters called Operational Taxonomic Units (OTU) and thus it can be built the distribution of sample abundance that indicates how many clusters contain r individuals, being $r = 1,2,3,..n$ (Hill et al. 2003).

To randomly select *n* individuals in a community that contains a finite and unknown number of species *S*, quantities $n_1, n_2, \ldots, n_r, \ldots, n_k$ are, in each case, the number of species that records *r* individuals among the *n* selected. So $\sum_{r=1}^{k} r n_r = n$. Clearly, n_0 represents the number of species not represented by any of the n chosen individuals and the number of species in the community is $S = \sum_{r=0}^{k} n_r$. If the number of species recorded by taking n individuals is called S_n, then $S_n = S - n_0$ (Chao and Shen. 2003). It is important to note that when adding a new individual to a sample, it may belong to an already present species or a new still unaccounted for.

2- The method of rarefaction curves allows estimating the richness of a medium although it is generally applied to compare richness between two or more communities, because it relieves the problems arising from inadequate and usually unequal samples size (Hughes-Hellmann 2005) (Gotelli and Colwell. 2001). The basic idea of the rarefaction by individuals when trying to estimate the richness is that from taking larger samples is possible to capture an increasing number of different species. Given a community with an unknown number of individuals and a number S of different species also unknown, you can take samples of size n and determine S_n, which is the number of different species found in the sample. At each rearrangement that is carried out in the sample by examining each individual, the number of detected species will grow cumulatively according to the dotted curve in Figure 1. Once recorded i of the n individuals in each reordering, the expected theoretical value of $E(S_i)$ is approximated by the average of the S_i obtained for each of them. The curve formed by the points $(i, E(S_i))$, which is drawn in a continuous line in Figure 1, is called of rarefaction and can be used to estimate the S richness of the medium.

The rarefaction curve represents the average of all accumulation curves built for different resamplings. When the number $i=n$ of examined individuals is reached, any accumulation curve will have reached the number S_n of species present in the sample size and therefore $E(S_n) = S_n$. If n also results large enough, the rarefaction curve would tend asymptotically to the value of the richness population S. That is; to determine the horizontal asymptote of a rarefaction curve the sample size should be increased until $E(S_n)$ observes a steady behavior as n grows and in such circumstance the amount $E(S_n) = S_n$ approximates the population richness S.

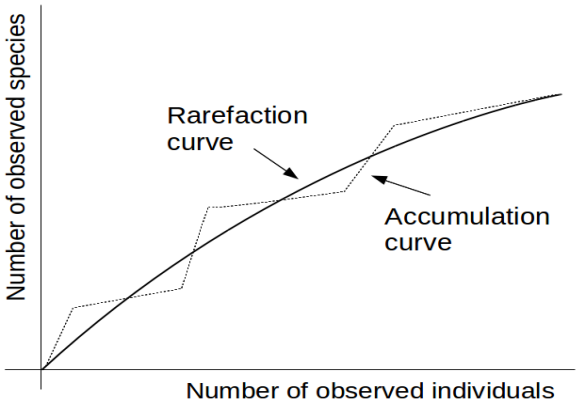


Figure 1. The line of points is any Accumulation Curve. The complete line is a Rarefaction Curve

3- An experimental model is built through a random process defined as follows:

Definition 1: Given a sample of size n, being, for each i with $i = 1,2,3,\ldots$, the random variable S_i that takes the values $S_i = S_{i-1}$ and $S_i = S_{i-1} + 1$ with the respective probabilities $1 - p_i$ and p_i being also $S_0 = S_n$. The succession of random variables $S_1, S_2, S_3, \ldots$ is hereinafter referred as Random Process of the Quantity of Species (Figure 2).

$$\ldots \quad S_{i-2} \quad S_{i-1} \quad S_i \quad S_{i+1} \quad S_{i+2} \quad \ldots$$

$$\ldots \quad i\text{-}2 \quad i\text{-}1 \quad i \quad i\text{+}1 \quad i\text{+}2 \quad \ldots$$

Figure 2. Random Process of the Quantity of Species.

The interpretation of the experimental model identifies S_i as the number of different species present in a sample of size $n+i$. In addition, p_i is interpreted as the probability of that by incorporating a new individual to a sample of size $n+i-1$, this one corresponds to a new species not present so far in the sample. It has been proved (Good 1953) that the probability that, once chosen n individuals, by selecting a new one it results to be from a species so far unaccounted for, this can be approximated by the quotient $T = \frac{n_1}{n}$ where n_1 is the number of species that appears once in the chosen sample and it must be assumed strictly greater than 0. This idea, provided by Turing, is used here to estimate p_i. Then, the proposed formula for calculating the probability of a new species associated with the random process of the quantity of species is $T_i = \frac{n°sgletons}{n+i-1}$ (Table 1). The number of singletons in each sample of size $n+i-1$ refers to the clusters formed by a single individual, when the sequences grouping process and the subsequent identification of each cluster with a different species are performed.

Table 1. State Probability

State	State Probability
$S_i=S_{i-1}$	$p_i = 1 - T_i$
$S_i=S_{i-1}+1$	$p_i = T_i$

Thus the expected value of S_i that would correspond to the rarefaction curve when i simulated individuals have been added to the sample is:

$E(S_i) = S_{i-1}(1-T_i) + (S_{i.-1}+1)T_i$ and operating $E(S_i) = S_{i-1} + T_i$ is obtained.

Figure 3 shows how the probability of new species tends to zero, $T_i \to 0$, as it grows the number of individuals incorporated to the sample by the simulation according to the experimental model. As the desired number of species S is finite (Good 1953), the rarefaction curve $E(S_i)$ should be reaching an asymptote that estimates it.

The value of T in each iteration must be considered to be an estimation of the probability of a new species and not exactly this probability. Therefore, the expected value $E(S_i)$ will be only an approximate estimation of S. Thus the built model will be appropriate if the computational experience shows a good performance in the evaluation of S. In the practice, with the idea of reducing the variance, the same calculation from the same sample can be realized several times and it might average the quantities estimated.

4- The Algorithm for Quantifying Species (AQS) is based on the Random Process of the Quantity of Species defined above. This algorithm performs the simulation by the Monte Carlo technique. Given the n size of the original sample, the value of the probability of new species estimator is determined. This value allows constituting the $[0, T_n]$ and $(T_n, 1]$ intervals so that when choosing a random number r such that $0 \le r \le 1$, if it falls within the first interval the new simulated individual belongs to a new species, and if it falls within the second interval it is a specimen of a known species. If the former occurs, the number of species in the medium is increased in 1 and if not, the existing proportions of each species are used to assign, by a new random number, the already known species to which the new individual belongs. Thus individuals are added until the account of the new species reaches a stable value or until it meets another cutting criterion of the simulation. The procedural steps are synthesized in a sequenced form below.

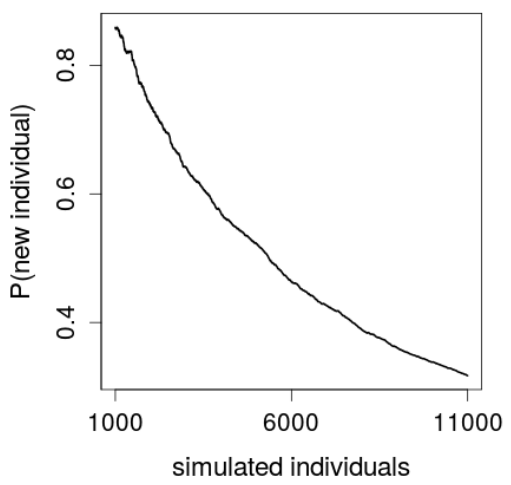


Figure 3. Decreased Probability of New Species

i- Given the chosen sample of size n, and its grouping in OTUs, the initial value of the Turing estimator $T_{i+1} = \frac{f_1}{i}$ is determined, being $i = n$ and f_1 the current number of singletons.

ii- A random number r is chosen, such that $0 \le r \le 1$, asking if it is in the $[0, T_{i+1}]$ interval. If so, $S_{i+1} = S_i + 1$ is performed and goes to step iv. If the opposite happens, $S_{i+1} = S_i$ is performed and goes to step iii.

iii- The distribution of the sample abundance is used to calculate the proportion of individuals in OTUs of $1,2,..n$ individuals and with these proportions it can be determined, by drawing lots according to them, to what group of already known OTUs the new simulated individual belongs. To establish to which specific OTU, among this group, belongs the new individual, a new drawing lot is performed with uniform probability for each OTU of the group.

iv- Let the new individual be or not of a new species, the sample now has one more element. It can be asked then whether the procedure should be cut because the chosen criterion is met, in which case the simulation would be complete. If the cutoff criterion is not met, then $i \leftarrow i + 1$ is assigned, the new distribution of abundance and the new estimate of Turing according to $T_{i+1} = \frac{f_1}{i}$ are calculated and it repeats from step ii.

The corresponding computer program was developed in R language and presented as Annex.

5- It is not possible to have sequences of 16S rRNA gene corresponding to all individuals of a real microbial community because, besides technological or economic constraints, the test would be destructive of the community itself. Therefore, to test the AQS procedure on an entire population, it can simulate it or work with a sample for which the rarefaction curve reaches the asymptote that estimates the quantity of species. In both cases it can then give by known the actual number of species and compare with the estimate obtained by applying the AQS procedure. This also allows comparing the performance of AQS with other forms of estimation.

With that idea a simulated community was built using the log series $\alpha x, \frac{\alpha x^2}{2}, \frac{\alpha x^3}{3}, ..., \frac{\alpha x^m}{m}$, which models the expected amount of species that are represented by $1, 2, ..., m$ individuals in the population. If N is the population size and S the number of species contained in it, the relations $\frac{S}{N} = [(1-x)/x][-\ln(1-x)]$ and $\alpha = \frac{N(1-x)}{x}$ are met (Fischer et al. 1943). According to this, values for S and N can be calculated from α and x amounts of ecological significance. Then, the values $\alpha = 5000$ and $x = 0.995$ were used (Magurran 2004) to obtain a community composed by 898341 individuals distributed among 26332 species.

On the other hand, to work also on real data, the water sample of deep sea FS396.archaea was considered (Haegeman et al. 2013) (Huber et al. 2007), this sample is composed of 16316 sequences of 16S rRNA and its rarefaction curve reaches an asymptotic behavior at that size. The number of species present in this sample, which for this experience it was considered as a whole community, is 346.

In both cases, the observed number of species was compared with the estimations obtained by the non-parametric statistics CHAO and ACE and these in turn with the estimations performed by AQS. For the sample obtained from the simulated population the development of the estimation was studied while the amount of simulated individuals increased and the Turing estimator value decreased. For the FS396.archaea sample five subsamples of six different sizes were taken and with each one of them 10 runs of the algorithm were performed in order to estimate the expected value $E(S_i)$ which in turn estimates S. To the comparison of performance the evaluation of the extrapolation was added by means of

$$\hat{S}_{n+m} \approx S_n + \hat{f}_0\left[1-\left(1-\frac{f_1}{n\hat{f}_0+f_1}\right)^m\right] \approx S_n + \hat{f}_0\left[1-\exp\left(\frac{-mf_1}{n\hat{f}_0+f_1}\right)\right]$$

(Chao et al. 2014).

Here, S_n is the richness of the sample of initial size n which is calculated from the same, f_1 is the number of singletons in the sample of size n and $\hat{f}_0$ is an estimation of the number of species not observed in the sample of size n. The ACE estimation was taken as $\hat{f}_0$ value that actually is designed as an estimator of the total species in the community and not as an estimator of the missing species in the sample. Thus this last amount may even be overvalued. The m value is the number of individuals that are ideally added to extrapolate. In the case where m become very large ($m \to \infty$), $\hat{S}_{n+m} \approx S_n + \hat{f}_0$ will result, so such value could be taken as an upper bound of the extrapolation that, at first, it also will be considered in the analysis. Finally an evaluation of the respective coefficients of variation was performed.

3 RESULTS

For the simulated population built with the Fischer distribution of parameters $\alpha=5000$ and $x=0.995$, the experimental test consisted of applying the AQS on an initial sample of 1000 individuals performing different numbers of iterations. In each run the value that reached the probability of a new species T_i was also obtained by cutting the simulation. Table 2 shows the results.

Table 2. AQS estimation of the community richness depending on quantity iterations

Based on the same sample of 1000 simulated individuals the performance of non-parametric estimators CHAO and ACE was compared with the AQS estimations obtained for increasing amounts of iterations. In turn, all of these results were compared with the actual number of species. Table 3 evidences the improvements that AQS produces in the estimation of richness regarding CHAO and ACE as the number of iterations increases. The series obtained searches an asymptotic value that not overtake S.

Table 3. Richness Estimation Comparative Performance

Reachness	Real	CHAO	ACE	AQS/ 45000	AQS/ 60000	AQS/ 200000	AQS/ 500000
N° of species	26332	6699	6751	12821	14615	19271	24559
%	100	25	26	49	56	73	93

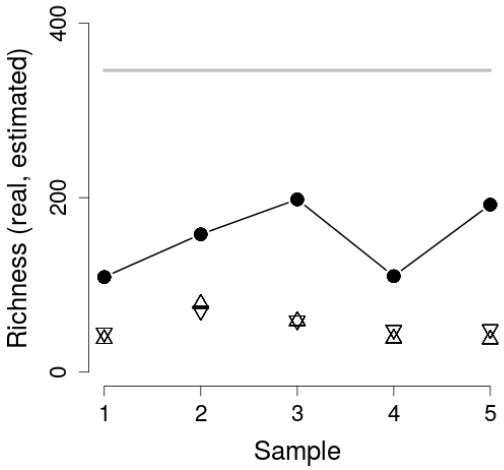


Figure 4. Richness Estimation (sample size 160). The thick gray line is the real richness. The black circles are the AQS estimations, the triangles are the CHAO estimations and the invert triangles are the ACE estimations.

By using the set FS396.archaea to test the performance of AQS the estimation based on five samples of 160 individuals representing approximately 1% out of 16316 individuals within the community, is former studied. In each case, 10 runs of the algorithm were performed by comparing the AQS average estimations with the CHAO and ACE estimations, with the extrapolation and its bound, and the real value of the richness $S = 346$. Figure 4 shows these results.

Then, five samples of each of the sizes 320, 640, 960, 1280 and 1600 were taken. On each sample 10 runs of the algorithm were performed establishing as estimation the average of the estimations for each run. The intention was to compare the estimations ACE and AQS with the actual known richness. Figure 5 shows the results obtained in each run for each sample and sample size.

Iterations	60000	200000	500000	897541
Species	14615	19271	24559	25327
Probability of new species	0.091	0.026	0.011	0.005

Table 4. AQS vs. ACE comparison as percent of real richness by relative size sample.

Sample Size	Real Richness	AQS (%)	ACE (%)
1%	100	44	16
2%	100	45	36
4%	100	64	47
6%	100	69	43
8%	100	78	71
10%	100	91	74

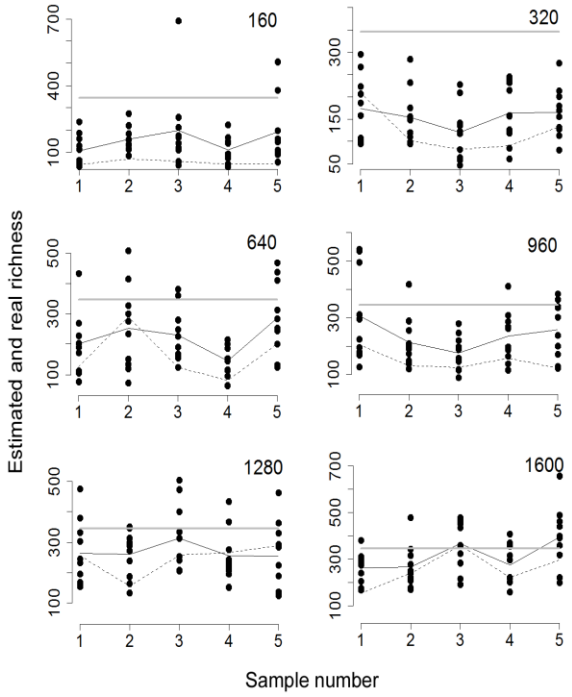


Figure 5. Comparison between AQS and ACE estimations. The horizontal gray line is the real richness. The complete line is the AQS estimation and the dashed line is the ACE estimation. The number in the upper right corners is the sample size.

Although, in ecological practice, one community sample can be chosen for each sample size, each of the estimators was averaged in order to compare them in statistics terms. Thus, Figure 6 shows that the increase in sample size produces, as expected, a better average performance of all of them. Particularly, when the results are observed in percentage of richness obtained by AQS and ACE according to the percentage sample size with reference to the population, AQS shows one improvement in the estimation as shown in Table 4. Differences in percentage estimations in favor of AQS over 100% of the richness are, for the present case, those of Table 5. Even for very large percentage-wise sample sizes, it is noted that the AQS estimation produces an improvement over the non-parametric statistical ACE.

Table 5. AQS Estimation Percentage Improvement

Sample Size (%)	1	2	4	6	8	10
AQS Improvement (%)	28	21	33	32	7	17

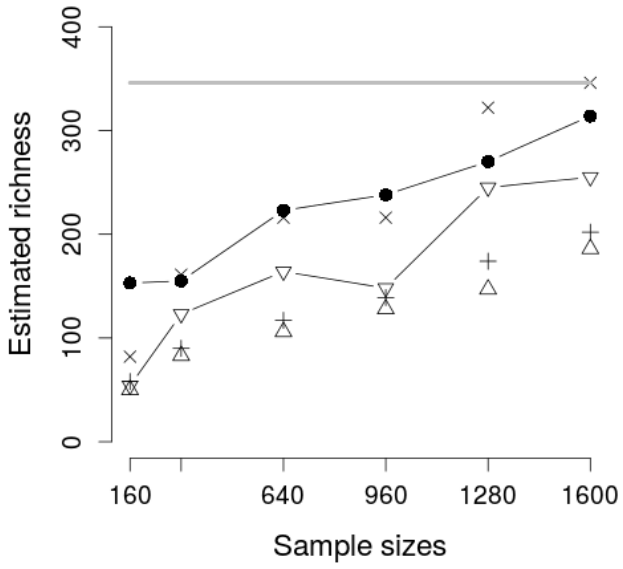


Figure 6. Richness average estimation by size. The thick gray line is the real richness. The black circles are the AQS estimations, the triangles are the CHAO estimations and the invert triangles are the ACE estimations. The symbols x and + are the extrapolation ACE estimation and the bound extrapolation ACE estimation respectively.

A study of the variability was additionally carried out by measuring the coefficient of variation to establish the stability of each one of the estimators obtained from the five samples of each sample size. As an AQS estimation the average obtained based on the 10 runs performed for each sample was taken. CHAO, ACE, Extrapolation and Bound Extrapolation values from each sample were considered and, as observed values, the amounts of species obtained in each one of the five samples for each size were taken. These latter values will show then the variability of the sample richness. Figure 7 shows these results. A detailed analysis of the results was performed in the following section.

4 DISCUSSION

Using Fischer distribution to build a simulated population is based on the observation, particularly in certain marine biospheres, of statistically rare phylotypes that fit in this distribution (Galand et al. 2009). The simulation of the population based on such distribution requires to choose α and x parameters that intervene in the calculation. The maximum number of individuals appearing in at least one cluster grows as the population size grows and potentially if an infinite population is assumed, it could be accepted that $m \to \infty$. In this case the amount of species is $S = \sum_{k=1}^{\infty} \frac{\alpha}{k} x^k = -\alpha \ln(1-x)$ and the total number of individuals is $N = \sum_{k=1}^{\infty} k \frac{\alpha x^k}{k} = \frac{\alpha x}{1-x}$. To deduce both

formulas it has been taken into account that $0 \prec x \prec 1$ (Fischer et al. 1943). Furthermore, the amount α is an intrinsic parameter of the richness

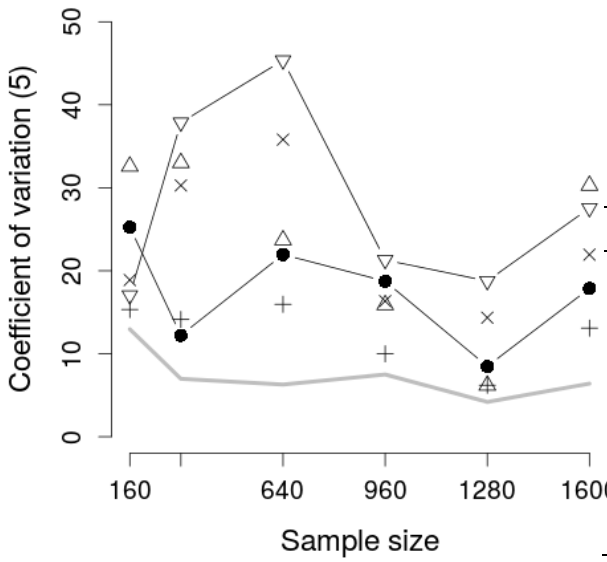


Figure 7. Comparison of coefficients of variation. The thick gray line is the real richness. The black circles are the AQS estimations, the triangles are the CHAO estimations and the invert triangles are the ACE estimations. The symbols x and + are the extrapolation ACE estimation and the bound extrapolation ACE estimation respectively.

of each population and can be used as an index of diversity (Magurran 2004). If values for the number of species S and N population size are established in advance, the value of x can be determined solving, by iterative methods, the equation $\frac{S}{N} = [(1-x)/x][-\ln(1-x)]$ that expresses the ratio between the number of species and the population size. The parameter x then depends on this reason and in the practice $x \succ 0.9$ is met without, of course, it can exceed the value 1 (Magurran 2004). Table 6 gives the values of the ratio $\frac{S}{N}$ calculated with different values of x assuming that the population size is $N = 1000000$. Thence, is possible to obtain the number of species S and with it, the value of the α parameter that can be calculated from N and x according to $\alpha = \frac{N(1-x)}{x}$. In all cases it is shown that $\alpha \prec S$. As αx is the first term of the log series, if x is close to 1, results approximately the expected number of species that appear once in the population, that is the minimum amount of species that can be considered rare. Also, when the ratio $\frac{S}{N}$ between the number of species and the population size decreases, as in some sense it expresses richness, α also does it. In the practice the richness index α can only be calculated from samples and in that case its value can be considered independent of the sample size when it has more than 1000 individuals. (Magurran 2004).

Table 6. Relation between x and α parameters with size and richness community

X	S/N	S	α
0.9	0.2558 ≈ 1/4	255817	111111.11
0.95	0.1577 ≈ 1/6	157700	52631.58
0.975	0.0946 ≈ 1/11	94600	25641.03
0.99	0.0465 ≈ 1/22	46500	10101.01
0.995	0.0266 ≈ 1/38	26600	5025.13
0.9975	0.0150 ≈ 1/67	15000	2506.27
0.999	0.0069 ≈ 1/145	6900	1001

If an initial approximation of richness between 5000 and 30000 species is considered for a population of 1 million individuals, the corresponding values x and α should range between $0.995 \le x \le 0.999$ and $1000 \le \alpha \le 5000$ respectively. For the chosen parameters $\alpha = 5000$ and $x = 0.995$, the value $\alpha x = 5000 \times 0.995 = 4975$ represents approximately the number of species of greatest rarity of the community. These species represent 19% of the total. In turn, according to the simulated distribution, there would be 10% of species represented by two individuals, 6% with three individuals in the population and so on. In short; a significant proportion of rare species in the community is present.

When a sample of 1000 individuals out of a total community of 897,541 was selected, the results dumped in Table 2 allowed checking the growing effectiveness of the AQS estimation while the number of individuals added in a simulated way to the sample increased. Also here, the decrease to 0 of the Turing estimation of a new species could be seen. Table 3 shows the proportion of the real richness detected as the number of iterations increases. Upon reaching the 500000, the estimated percentage of the richness by AQS is 93%, which is a qualitatively different performance than the one provided by ACE (26%) and CHAO (25%) from the same sample.

To select the F396.archaea sample and use it as a test population its relatively large size was considered and the asymptotic form which, considered all individuals, presents its rarefaction curve (Haegeman et al. 2013). By selecting five samples out of 160 individuals the AQS estimation was higher than that obtained as a bound of extrapolation and almost double the estimates made by ACE, CHAO and its

own extrapolation when a number of ideal individuals who tripled the sample size were added. $m = 3n$ is the maximum recommended value by adopting a statistical point of view to perform the extrapolation (Chao et al. 2014). However, the experience also allowed confirming that the best estimation provided by AQS barely reached half of the real richness. In this sense, each estimation was averaged considering the five samples to obtain a graph that illustrates the percentage of the estimated real richness per statistical. Table 7 shows marked differences between the average estimations and the actual value.

Table 7. Estimated richness as percent of real richness.

	Estimated Richness (%)
Real Richness	100
AQS	44
Extrapolation Bound	24
ACE Extrapolation	17
ACE	16
CHAO	14

The increase in sample size allowed reducing these differences. Figure 5 shows that for samples of 1600 individuals, about 10% of the population size, cases of very precise estimation and even overestimation occurred. In general terms, Figure 6 and table 4 confirm the improved performance of AQS compared to the non-parametric estimator ACE . An extrapolation behavior with $m = 3n$ similar to that of CHAO is also observed producing both underestimation of the richness. The extrapolation bound is closer to the values obtained by AQS, especially when the sample size grows. However, for the smaller sample size AQS provides a better estimation. As expected, in addition, a more precise estimation occurs when the size of the original sample increases.

In spite of the analysis performed, the results always evidence the differences in the precision of the estimation obtained in the simulated case by the Fischer distribution and the real case analyzed, suggesting the study of other measures to account for the community diversity, especially in the case in which the analyses should be performed from a single sample.

Figure 7 analyzes the variability of different estimations. First, it should be pointed out the inherent variability of sampling. This can be appreciated by varying the amount of different species that appear in different samples. These are represented by a gray line and exhibit the lowest relative variation for all sample sizes. According to this, the increase of the relative variation for each type of estimation will be linked to the second source of variability that is the inherent to the method used to establish it. Non-parametric estimations reveal percentages of variation between 6% and 33% for CHAO and between 17% and 45% for ACE, with respective variation ranges of 27% and 28%. AQS reveals lower rates, between 8% and 25%, and in turn lower range of variation of the same, 17%. Thus the AQS estimation better appreciates the richness and with less variability.

5 CONCLUSIONS

- An alternative approach has been presented based on an experimental model that "fits" the enlargement process of the sample needed to better accurate the richness estimation. Such approach, related to the data mining, uses an initial subset to predict a parameter value of the community.
- The results have significantly improved the estimations of richness carried out based on models and assumptions of analytical and statistical type.
- Anyway, is once again clear the need to assess the biodiversity through various measures supplementing the knowledge of the community, while providing a more reliable assessment of their properties.
- Given the best, though still insufficient, performance achieved in the richness estimation according to the alternative approach proposed, its application to other measures that quantify the amount of information and the distribution of species in the community is suggested.

REFERENCES

Armougom, F. and Raoult, D. 2009. Exploring Microbial Diversity Using 16S rRNA High-Throughput Methods. Journal of Computer Science & Systems Biology Volume 2(1): 074-092 (2009) – 074

Chao, A. 1984. Nonparametric estimation of the number of classes in a population. Scand J Statist 11: 265-270.

Chao, A and Lee, S. 1992. Estimating the Number of Classes via Sample Coverage. Journal of American Statistical Association. Volume 87. Issue 417.

Chao, A and Shen, T. 2003 Nonparametric estimation of Shannon's index of diversity when there are unseen species in sample. Environmental and Ecological Statistics 10, 429-443.

Chao, A. Gotelli, N. J. Hsieh, T.C. Sander, E. L. Ma, K.H. Colwell, R. K. and Ellison A. M. 2014. Rarefaction and extrapolation with Hill numbers: a framework for sampling and estimation in species diversity studies. Ecological Monographs. 84 (1) 2014 pp.45-67.

Fischer, R. Corbett, S. and Williams, C. 1943. The Relation Between the Number of Species and the Number of Individuals in a Random Sample of an Animal Population. The Journal of Animal Ecology. British Ecological Society. Vol 12 N°1 (1943)

Galand, P. E. Casamayor, E. O. Kirchman, D. L. and Lovejoy, C. 2009. Ecology of the rare microbial biosphere of the Artic Ocean. PNAS. Vol. 106 no. 52 22427-22432.

Good, I. 1953. "The Population Frequencies of Species and Estimation of Population Parameters". Biometrika. Vol 40 N° 3/4.

Gotelli, N and Colwell,R. 2001. Quantifying biodiversity: procedures and pitfalls in measurement and comparison of species richness. Ecology Letters. 4: 379-391

Haegeman, B. Hamelin, J. Motitarty, J. Neal, P. Dushoff, J. and Weitz, J. 2013 Robust estimation of microbial diversity in theory and in practice. ISME Journal 2013, 1-10

Hill, M. O. 1973. Diversity and Eveness: A Unifying Notation and Its Consequeces. Ecology. Vol.54. No 2 pp 427-432.

Hill, T. Walsh, K. Harris, J. and Moffett, B. 2003. Using Ecological Diversity Measures with Bacterials Communities. FEMS.Microbiology Ecology 43 1-11

Hillis, D. M. Moritz, C. and Mable, B. K. 1996. Molecular Systematics. Second Edition, Sinauer Associates, Inc. Publishers. Sunderland, MA. USA.

Huber, J. A. Mark Welch, D. B. Morrison, H. G. Huse, S. M. Neal, P.R. Butterfield, D. and A. Sogin, M. L. 2007. Microbial Population Structures in the Deep Marine Biosphera. Science 318, 97(2007) DOI: 10.1126/science.1146689

Hughes, J. Hellmann, J. Ricketts, T. and Bohannan, B. 2001. Counting the uncountable: statistical approaches to estimating microbial diversity. Applied and Environmental Microbiology. 4399-4406.

Hughes,J and Hellman, J. 2005. The Application of Rarefaction Techniques to Molecular Inventories of Microbial Diversity. Methods in Enzymology. Vol 397.

Huse, S. M. Welch, D.M. Morrison, H.G. and Sogin, M.L. 2010. Ironing out the wrinkles in the rare biosphere through improved OTU clustering Environmental Microbiology (2010) 12(7), 1889–1898

Magurran, A. 2004. Measuring Biological Diversity. Blackwell Science Ltd.

Magurran, A and McGill, B.J. 2011. Biological Diversity. Oxford University Press.

Nádas, A. 1985. On Turing's Formula for Word Probabilities. IEEE Transactions on Acoustics, Speech and Signal Processing. Vol ASSP-33 N° 6.

O'Hara, R. 2005. Species richness estimators: how many species can dance on de head of a pin. Journal of Animal Ecology. 74, 375.386

Roesch, L. Fulthorpe, R. Riva, A. Casella, G. Hadwin, A. Kent, A. Daroub, S. Camargo, F. Farmerie, W. and Triplett, E. 2007. Pyrosequencing enumerates and contrasts soil microbial diversity. The ISME Journal. 1, 283-290.

Schloss, P and Handelsman, J. 2005. Introducing DOTUR, a Computer Program for Defining Operational Taxonomic Units and Estimating Species Richness. Applies and Environmental Microbiology. pp 1501-1506