

Índice

1. ¿Qué es *de novo*?
 2. Datos de ejemplo
 3. Software STACKS
 - a. *process_radtags*
 - b. *ustacks*
 - c. *sstacks*
 - d. *cstacks*
 - e. *tsv2bam*
 - f. *gstacks*
 - g. *populations*
-

Que és *de novo*

De novo es una forma de ensamblaje del genoma sin la ayuda de datos genómicos de referencia, así es, un ensamblaje inicial de un genoma. Esto asume que no se conoce detalles del genoma, como el tamaño y la composición de las secuencias del ADN. Por esto, son bastante utilizados en organismos no-modelos donde no hay disponible genomas para muchas especies. En la ausencia de un genoma referencia, los loci no se pueden visualizar posicionalmente; en su lugar, cada locus se examina de forma independiente para comprobar si es un valor atípico para detectar falsos positivos. En la figura abajo (Fig. 1) una comparación de (a) *de novo* y (b) genoma referencia.

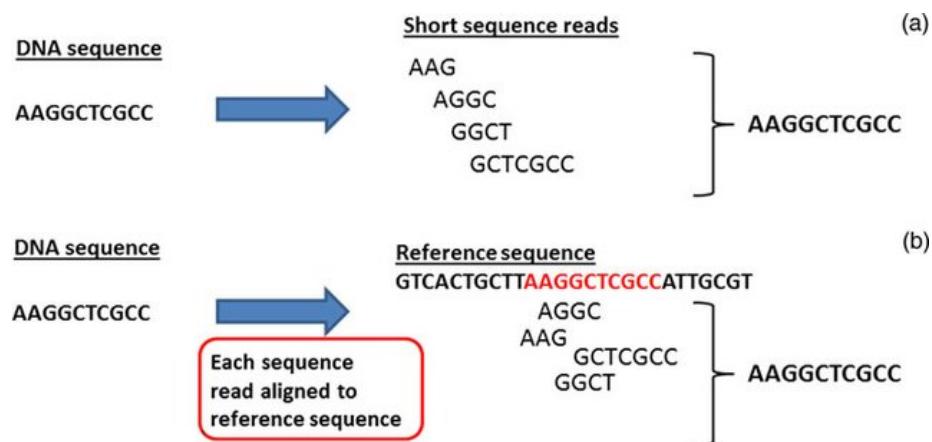


Figure 1: Comparación de (a) *de novo* y (b) genoma referencia (Fuente).

Además, *de novo* puede ser utilizado si hay un genoma referencia pero no es de alta calidad. Un análisis *de novo* complementario es bueno para comparar cuántos

loci pueden faltar en el genoma de referencia o para identificar loci atípicos que no están en la referencia ensamblada. Cuando el genoma de referencia está distante, o si su calidad es cuestionable, tiene sentido adoptar un enfoque híbrido: ensamblar loci de novo y luego alinear sus secuencias de consenso y respaldar la información posicional en el conjunto de datos de novo. Esto permite aprovechar el genoma de referencia sin comprometer la consistencia de las llamadas de genotipo. Este enfoque híbrido permite una comparación directa de los análisis de novo y basados en referencias. Cuando el genoma es de buena calidad y las propiedades genómicas (por ejemplo, contenido repetido) del sistema son tales que el ensamblaje de novo de los loci RAD funciona bien, los dos enfoques deberían producir resultados muy similares.

En este curso vamos a utilizar STACKS para hacer la ensamblaje *de novo*. Pero es importante saber que hay varios otros softwares que lo hacen y el uso de uno u otro dependerá del tipo de datos y el propósito del estudio. Por ejemplo, ipyrad es bastante utilizado para filogenómica, SOAPdenovo es un software nuevo y muchos otros.

Datos de ejemplo

Lupinus (Contreras-Ortiz, et al., 2018)

Lupinus es un género de planta con una diversificación muy grande en los Andes, con 85 especies. Algunas especies cambian su historia de vida anual a perenne, esto ha sido sugerido como una adaptación clave que facilita la colonización de hábitats montanos (3500-4900m). Aquí vamos a utilizar muestras de dos especies estrechamente relacionadas de *Lupinus* (*L. triananus* (A) y *L. alopecuroides* (B) en la figura abajo).



Figure 2: Las dos especies de *Lupinus* que vamos trabajar: (A) *L. triananus* y (B) *L. alopecuroides* (foto de Contreras-Ortiz, et al., 2018).

L. alopecuroides se distribuye en la cordillera Central y Oriental al sur y *L. triananus* en la cordillera Oriental (ver mapa abajo). Vamos a utilizar cinco

individuos de cada especie que fueron secuenciados a través de la técnica nextRAD en el trabajo de Contreras-Ortiz, et al., 2018 y que están disponibles públicamente. El secuenciamiento fue realizado en Illumina NextSeq500 con lecturas de 150bp y *single-end*.

Artículos con más información sobre el conjunto de datos:

- Contreras-Ortiz, et al., 2018; - Nevado et al., 2018.
-

Software STACKS



[Website](#) | [Manual](#)

Stacks es un “*pipeline*” para construir loci a partir de secuencias cortas, como las generadas en la plataforma Illumina. Stacks se desarrolló para trabajar con datos basados en enzimas de restricción, como RAD-seq, con el propósito de construir mapas genéticos y realizar genómica y filogeografía de poblaciones.

Stacks identifica los loci en un conjunto de individuos, usando *de novo* o alineados con un genoma de referencia, y luego genotipa cada locus. Un análisis *de novo* en Stacks se desarrolla en seis etapas principales (Fig. abajo). Primero, las lecturas son demultiplexadas y limpiadas por el programa **process_radtags**. Las siguientes tres etapas comprenden la canalización principal de Stacks: construcción de loci (**ustacks**), creación del catálogo de loci (**cstacks**) y comparación con el catálogo (**sstacks**). En la quinta etapa, se ejecuta el programa **gstacks** para ensamblar y fusionar contigs de extremos emparejados, llamar a sitios variantes en la población y genotipos en cada muestra. En la etapa final, se ejecuta el programa de **populations**, que puede filtrar datos, calcular estadísticas de genética de poblaciones y exportar una variedad de formatos de datos.

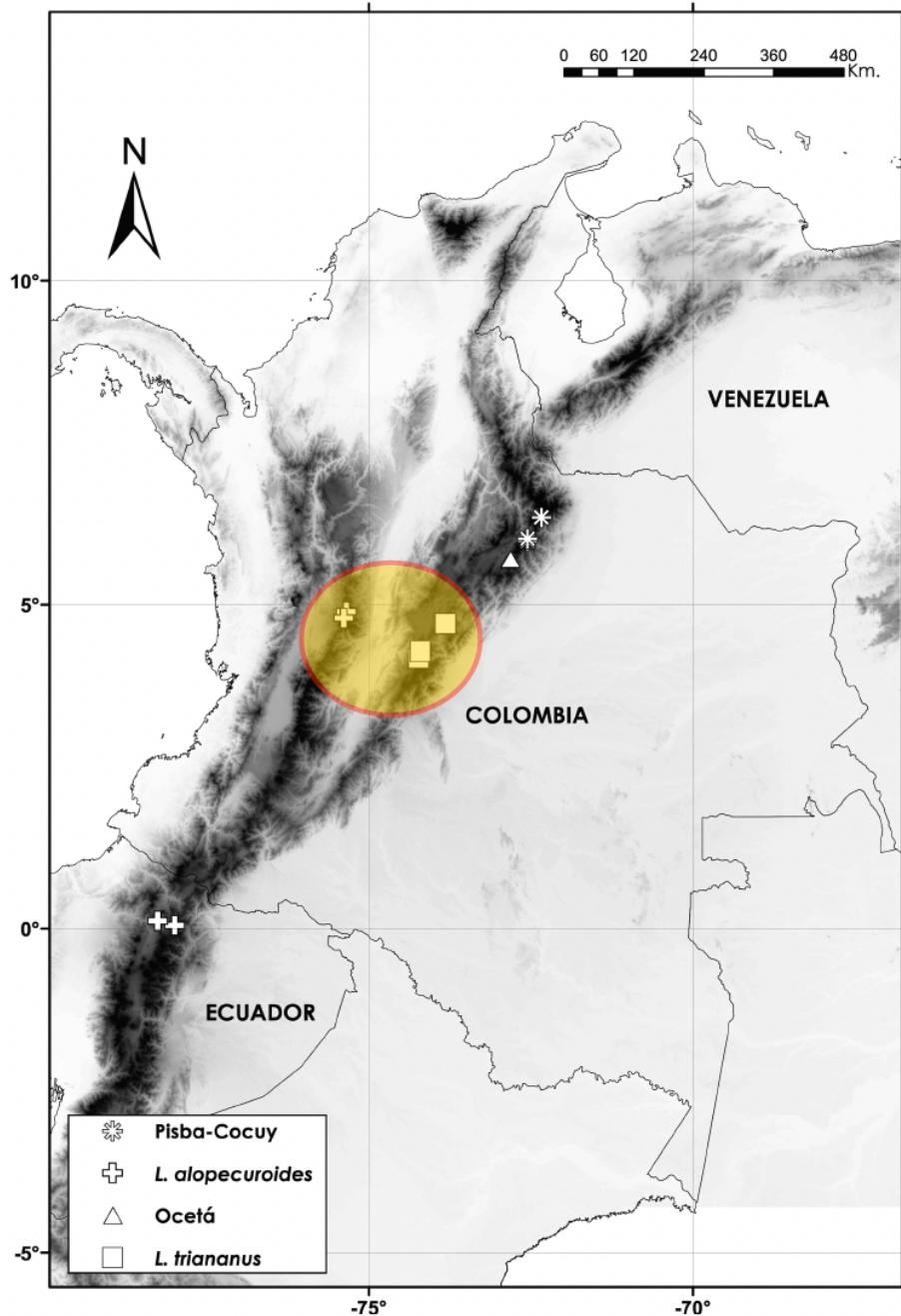


Figure 3: Mapa con la distribución de las dos especies de *Lupinus* que vamos a utilizar en nuestro dataset - en amarillo (foto modificada de Contreras-Ortiz, et al., 2018).

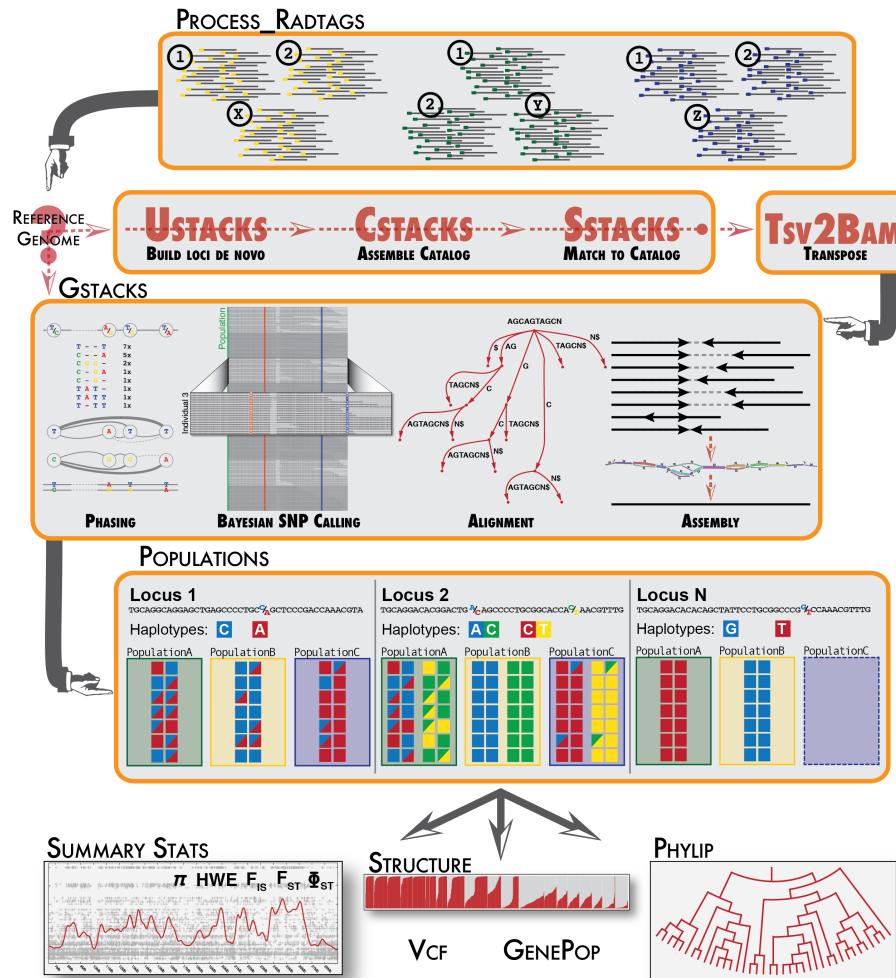


Figure 4: Pasos realizados en Stacks (Foto de Stacks).

Antes de empezar:

Vamos a copiar los datos brutos que analizaremos a su usuario. Los datos **all.fastq.gz** se encuentran en la carpeta compartida **xxxx**. Ejecute el comando **scp** para copiar los datos a tu usuario.

Además, creamos una carpeta donde vamos poner todos los resultados de Stacks. Entonces en tu directorio y usando el comando **mkdir** generemos la carpeta llamada **01_stacks**. Ahora estamos listos para hacer el primero paso del pipeline :)

Además, en el clúster para cargar Stacks necesitamos usar este comando en el sbatch o en la línea de comando: **module load stacks/2.59**

process_radtags

En un análisis típico, los datos estarán en formato FASTQ, que es el formato que normalmente se recibe de un secuenciador de Next-gen, como el Illumina. El primer requisito es demultiplexar los datos sin procesar para recuperar las muestras individuales. Al hacer esto, usaremos *Phred scores* proporcionadas en los archivos FASTQ para descartar las lecturas de secuenciación de baja calidad. Estas tareas se realizan en *process_radtags*.

Process_radtags examina las lecturas crudas de Illumina y, en primer lugar, comprueba que los barcodes y el sitio de corte RAD (Fig. abajo) estén intactos y demultiplexa los datos. Si hay errores en el barcode o en el sitio RAD dentro de un cierto margen, *process_radtags* puede corregirlos. En segundo lugar, desliza una ventana a lo largo de la lectura y verifica el puntaje de calidad promedio dentro de la ventana. Si la puntuación cae por debajo del 90% de probabilidad de ser correcta (una puntuación *phred* bruta de 10), la lectura se descarta. Estos procedimientos eliminan las lecturas en que la secuencia se degrada a medida que se secuencia, pero aún admite algunos errores de secuenciación que se puede corregir.

Archivos necesarios: - archivo fastq.gz con todas las muestras crudas (no procesado) - archivo *.txt* con dos columnas separadas por un *tab*. La primera columna debe corresponder al barcode y la segunda al nombre de la muestra - la información sobre los barcodes y nombres está disponible en la tabla “Info_data_Lupinus.csv” que compartimos en GitHub.

Parámetros: Muchos de los parámetros utilizados aquí dependerán de cómo se haya secuenciado la biblioteca. Por lo tanto, es importante tener en cuenta que estos cambiarán de una biblioteca a otra según el conjunto de barcodes, adaptadores y el protocolo de secuencia utilizado.

Hoy usaremos estas opciones (para otras consulte el manual de Stacks):

-p = ruta al directorio donde esta el archivo fastq.gz (sin el nombre del archivo)

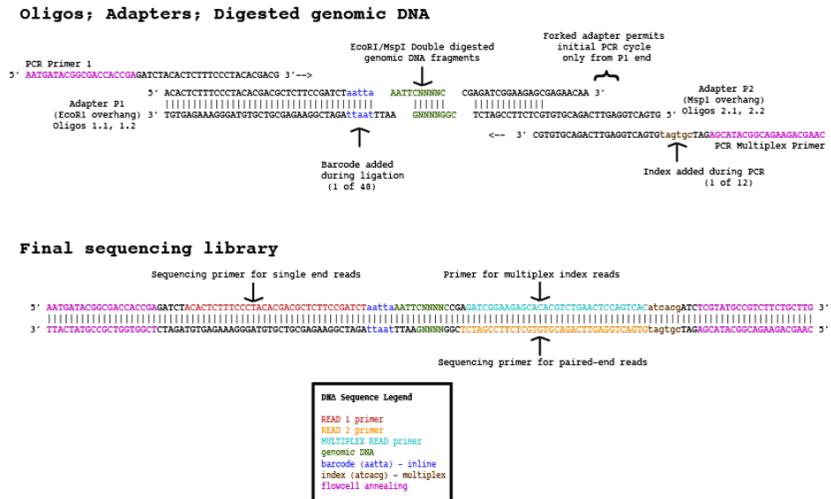


Figure 5: Diagrama con un ejemplo de los elementos (barcode, adaptadores, ...) de cada fragmento de ADN secuenciado (Foto de Peterson et al., 2012)

-i = tipo de archivo de entrada, ya sea ‘fastq’, ‘gzfastq’ (fastq comprimido con gzip), ‘bam’ o ‘bustard’
 -b = ruta a un archivo que contiene el listado de los barcodes en la primera columna y en la segunda el nombre de la muestra correspondiente (mirar “Archivos necesarios” para más info)
 -o = ruta donde se van a poner los archivos procesados (01_stacks)
 -q, --quality = descartar lecturas con baja calidad
 -E = especificar cómo se codifican las puntuaciones de calidad, ‘phred33’ (Illumina 1.8 + / Sanger, predeterminado) o ‘phred64’ (Illumina 1.3-1.5)
 -D = capturar lecturas descartadas en un archivo
 -e [enz], --renz_1 [enz] = la enzima de restricción utilizada (acá vamos utilizar ecoRI)
 --inline_null = barcode está en la misma línea que la secuencia, ocurre solo en lecturas *single-end* (predeterminado)
 --barcode_dist_1 = la cantidad de discrepancias permitidas al rescatar códigos de barras (predeterminado 1)
 --filter_illumina = descartar las lecturas que hayan sido marcadas por el filtro de pureza de Illumina como fallidas

Memoria necesaria: 10mb

Tiempo de ejecución: ~25min

Output: un archivo fq.gz por individuo (+discards y log) porque es *single-end*

ustacks

Ustacks alineará para cada muestra las secuencias cortas en alelos putativos. Al comparar estos, se formará un conjunto de loci putativos y detectará SNP en cada locus utilizando una metodología de maximum likelihood.

Archivos necesarios:

- un archivo fq.gz por muestra generado por *process_radtags*

Parámetros:

Tenga en cuenta que un comando *ustacks* debe realizarse para cada muestra.

-t = tipo de archivo de entrada. Tipos admitidos: fasta, fastq, gzfasta o gzfastq (predeterminado: adivinar)

-f = ruta del archivo de entrada con el nombre del archivo (.fz.gz)

-o = ruta donde escribir los resultados (**01_stacks**)

-i = un ID numérico único para cada muestra (1,2,3,...)

-m = profundidad mínima de cobertura requerida para crear un stack (predeterminado 3, usaremos 5)

-M = distancia máxima (en nucleótidos) permitida entre stacks (predeterminado 2)

-p = habilitar la ejecución paralela con número de subprocessos (usaremos -p 2)

--model_type = ‘snp’ (predeterminado), ‘bounded’(vamos usar esto), o ‘fixed’

Memoria necesaria: ~2.5gb

Tiempo de ejecución: ~12 horas (esto es el paso mas lento de todos!)

Output: 3 archivos por muestra: **.tags.tsv.gz**, **.snps.tsv.gz**, **.alleles.tsv.gz**

cstacks

Crea un catálogo a partir de cualquier conjunto de muestras procesadas por *ustacks*, generando un conjunto de loci de consenso fusionando alelos.

Archivos necesarios:

- los archivos de las muestras generado por *ustacks* - un archivo *.txt* con el **mapa de población** que tiene en cada línea: muestra población a que pertenece.

Parámetros:

-P, --in_path = directorio de entrada que tenga los archivos generados por Stacks hasta ahora (**01_stacks**).

-M, --popmap = ruta a un mapa de población (mirar archivos necesarios)

-p = habilitar la ejecución paralela con número de subprocessos (usaremos -p 2)

-n = número de discrepancias permitidas entre los loci putativos cuando se crea el catálogo (predeterminado 1, usaremos 2)

Memoria necesaria: ~5.5gb

Tiempo de ejecución: ~40min

Output: 3 archivos con el nombre “catalog” (.tags, .snps y .alleles)

sstacks

Los conjuntos de loci putativos construidos por *ustacks* se van a buscar en el catálogo producido por *cstacks*. Esto es, todas las muestras van ser comparadas con el catálogo.

Archivos necesarios: - catálogo generado en *cstacks* - archivos generados por *ustacks* - mapa de población (lo mismo que generamos para *cstacks*)

Parámetros:

- P, --in_path = directorio de entrada que tenga los archivos generados por Stacks hasta ahora (01_stacks).
- M, --popmap = ruta a un mapa de población
- p, --threads = habilitar la ejecución paralela con número de subprocessos (usaremos -p 2)

Memoria necesaria: ~5.2gb

Tiempo de ejecución: ~22min

Output: 1 archivo .matches.tsv.gz por muestra

tsv2bam

El programa *tsv2bam* transpondrá los datos para que estén orientados por locus, en lugar de por muestra.

Archivos necesarios:

- archivos generados por *ustacks* y *sstacks* - mapa de población (lo mismo que generamos para *cstacks*)

Parámetros:

- P, --in-dir = directorio de entrada que tenga los archivos generados por Stacks hasta ahora (01_stacks).
- M, --popmap = ruta a un mapa de población
- t — habilitar la ejecución paralela con número de subprocessos (predeterminado 1, usaremos 2)

Memoria necesaria: ~600mb

Tiempo de ejecución: ~4min

Output: archivo BAM estándar para cada muestra

gstacks

El programa gstacks examinará un conjunto de datos RAD un locus a la vez, observando a todos los individuos en la metapoblación para ese locus. Para los análisis *de novo*, gstacks empezará con los resultados del pipeline (ustacks → cstacks → sstacks → tsv2bam) y alinearán las lecturas de las muestras individuales con el locus. Al hacer esto, gstacks identificará SNP dentro de la metapoblación para cada locus y luego genotipará a cada individuo en cada SNP identificado. Una vez que se han identificado y genotipado los SNPs, gstacks dividirán los SNPs en cada locus, en cada individuo, en un conjunto de haplotipos.

Archivos necesarios:

- archivos *.matches.bam* creado por *tsv2bam* - mapa de población (lo mismo que generamos para cstacks*)

Parámetros:

- P = directorio de entrada que contiene archivos *.matches.bam creados por el pipeline *de novo* de Stacks (01_stacks).
- M, --popmap = ruta a un mapa de población

Memoria necesaria: ~350mb

Tiempo de ejecución: ~15min

Output: catalog.fa.gz que contiene la secuencia de consenso para cada locus ensamblado en los dato, y catalog.calls, que es un archivo personalizado que contiene datos de genotipado.

populations

Archivos necesarios: - los archivos generados por Stacks - mapa de población (lo mismo que generamos para cstacks)

Parámetros:

Lista completa de parámetros aquí.

- P, --in_path = directorio de entrada que tenga los archivos generados por Stacks (01_stacks).
- O, --out_path = ruta a un directorio donde escribir los archivos de salida. (predeterminado al valor de -P.)
- M, --popmap = ruta a un mapa de población (lo mismo que se usó antes)
- t, --threads = habilitar la ejecución paralela con número de subprocessos (predeterminado 1, usaremos 2)
- p, --min-populations [int] = número mínimo de poblaciones en las que debe estar presente un locus para ser procesado
- r, --min-samples-per-pop [float] = porcentaje mínimo de individuos en una población requerido para procesar un locus para esa población (usaremos 0.8)
- R, --min-samples-overall [float] = porcentaje mínimo de individuos en

poblaciones requerido para procesar un locus (usaremos 0.8)

--min-mac [int] = especificar el número mínimo de muestras que un alelo tiene que estar presente para procesar un SNP (aplicado a la metapoblación, usaremos 2)

--max-obs-het [float] = especificar la heterocigosidad máxima observada requerida para procesar un sitio de nucleótidos en un locus (aplicado a la metapoblación, usaremos 0.5)

--write-random-snp = restringir el análisis de datos a un SNP aleatorio por locus

--fstats = habilitar estadísticas basadas en SNPs y las F basadas en haplotipos

--vcf = output SNPs y haplotipos en Variant Call Format (VCF)

--structure = output los resultados en el formato de Structure

--plink = output genotipos en formato PLINK

Para este conjunto de datos, vamos correr *populations* dos veces. Uno usando todos los SNPs para calcular estadísticas sumarias --fstats y otro para generar los archivos para diferentes programas usando solamente un SNP aleatorio por loci --write-random-snp --vcf --structure --plink. Crear una carpeta para cada vez que ejecutaremos *populations*.

Memoria necesaria: ~1gb

Tiempo de ejecución: <1min

Output: muchos archivos - dependerá de los parámetros que elija.
